Addressing double dipping through selective inference and data thinning

Anna Neufeld

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:
Daniela Witten, Chair
Alex Luedtke
Ali Shojaie

Program Authorized to Offer Degree:
Statistics

University of Washington

Abstract

Addressing double dipping through
selective inference and data thinning

Anna Neufeld

Chair of the Supervisory Committee:
Daniela Witten
Departments of Statistics and Biostatistics

While classical statistical methods assume that we only ever test pre-specified hypotheses about pre-specified models, the reality is that scientists often explore their data before coming up with models and hypotheses of interest. We refer to the practice of using the same data to generate and then test a hypothesis, or to fit and then evaluate a model, as *double dipping*. Problems arise when standard statistical procedures for testing hypotheses or evaluating models are applied in settings that involve double dipping. To circumvent the challenges associated with double dipping, one can take two possible approaches. The first approach is to develop specialized statistical procedures that *account* for double dipping. The second approach is to *avoid* double dipping by conducting hypothesis generation and hypothesis testing (or model fitting and model evaluation) on independent datasets. When we only have access to one dataset, we typically accomplish this via *sample splitting*, in which we split the observations in our dataset into two smaller datasets, such that one can be used for hypothesis generation or model fitting, and the second can be used for hypothesis testing or model evaluation.

The first portion of this thesis proposes a *selective inference* framework for conducting inference after fitting a regression tree. *Selective inference* frameworks allow us to generate and test a null hypothesis using the same data by conditioning on the event that the data

led us to select a given null hypothesis.

The second portion of this thesis is motivated by problems that arise in the analysis of single-cell RNA sequencing data. In the analysis of single-cell RNA sequencing data, scientists often first use their data to estimate latent variables, and then wish to either evaluate these latent variable models or use these estimated latent variables for downstream inference. The pipelines used for latent variable estimation are very complex, and so developing specialized procures to avoid double dipping in this setting would be very difficult. Furthermore, these are unsupervised problems for which sample splitting is not an option: estimating latent variable coordinates for half of the observations does not yield latent variable coordinates for the remaining observations, and thus no downstream evaluation or inference can be performed. To address these challenges, we propose *Poisson count splitting*, which splits a single observation in a dataset into two components, which are independent under a Poisson assumption. We show that Poisson count splitting provides an alternative to sample splitting that allows us to avoid double dipping in unsupervised settings. As single-cell RNA sequencing data is often thought to be overdispersed relative to the Poisson distribution, we next propose *negative binomial count splitting*, which allows us to avoid double dipping under a more realistic and more general negative binomial assumption.

In the final portion of this thesis, we generalize the count splitting framework to a variety of distributions, and refer to the generalized framework as *data thinning*. Data thinning is a very general alternative to sample splitting that is useful far beyond the context of single-cell RNA sequencing data, and, unlike sample splitting, can be applied in both supervised and unsupervised settings.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Daniela Witten, for her enthusiastic and kind support, dedication, and mentorship throughout the course of my PhD. I could not have asked for a better advisor and role model. I would also like to thank Lucy Gao, who provided additional guidance and mentorship on all four projects that are included in this dissertation. I would also like to thank my additional collaborators on the work included in this dissertation: Joshua Popp, Alexis Battle, Ameer Dharamshi, Keshav Motwani, and Jacob Bien. I am also very grateful to Ali Shojaie, Alex Luedtke, and Cole Trapnell for making the time to be on my committee.

I would like to acknowledge the support of my family, and my wonderful undergraduate mentors who influenced me and supported me along my path to graduate school. In particular, I would like the thank my parents, my brother, Julie Blackwood, Brianna Heggeseth, and Andrea Danyluk.

Finally, I would like to sincerely thank past and present members of the UW (bio)statistics community for making Padelford and HRC very joyful places to be over the past several years. I am especially grateful for the homework group that got me through some very tough courses at UW, everyone who rode bikes or ate bagels with me, and all of my other friends who have made Seattle feel like home.

DEDICATION

To my friends and the mountains.

Chapter 1
INTRODUCTION

Classical statistical methods are designed for testing pre-specified hypotheses about pre-specified models. The reality of modern data analysis is that scientists often collect large and complex datasets. They then carry out an exploratory process that involves visualizing the data, fitting several models, evaluating these models to select their favorite, and then testing hypotheses about these models. We refer to the practice of using the same dataset for multiple tasks along this exploratory pipeline as *double dipping*. In this thesis, we focus on two specific examples of double dipping.

Example 1: Using the same data to generate and test a null hypothesis.

Example 2: Using the same data to fit and evaluate a model.

In Example 1, if standard statistical tests are applied to data that was used to generate a hypothesis, then these tests will not control the Type 1 error rate [Fithian et al., 2014, Taylor and Tibshirani, 2015]. In Example 2, the double-use of data will lead to downward bias of typical loss functions. This means that if we fit and evaluate several competing models with the purpose of selecting one with the lowest value of the loss function, we will select overly complex models that overfit the data.

In the case of Example 1, recent interest has focused on *selective inference*, a framework that enables a data analyst to generate and test a hypothesis on the same data [Taylor and Tibshirani, 2015, Fithian et al., 2014]. The main idea is to conduct inference using the conditional distribution of the data given the event that the data led us to generate this hypothesis. This framework has been applied to a number of problems, such as inference after variable selection in regression [Loftus and Taylor, 2015, Lee et al., 2016, Tibshirani

et al., 2016], inference after changepoint detection [Hyun et al., 2016, Jewell et al., 2022], and inference after clustering [Gao et al., 2022, Chen and Witten, 2023]. In Chapter 2 of this thesis, we develop a selective inference framework for testing for a difference in means between regions selected by a CART regression tree [Breiman et al., 1984] by conditioning on the event that the CART tree selected the given regions. CART regression trees are often lauded for being interpretable, but prior to our work, there was no way to rigorously quantify the statistical significance of a split in a given CART regression tree. Thus, our framework allows practitioners to ensure that they are interpreting signal, rather than noise, when interpreting a CART tree.

While the selective inference framework provides a valuable way to circumvent the problems associated with double dipping in the context of regression trees, there are other scenarios in which selective inference frameworks are too inflexible to be practical. A regression tree is an example of a hypothesis generation procedure that can be described algorithmically (i.e. we can analytically characterize the event that the CART algorithm selected a particular set of regions). Selective inference frameworks cannot be used for hypothesis generation procedures that cannot be described algorithmically (e.g. an expert observing a graph). Furthermore, existing selective inference frameworks nearly always assume that the data are Gaussian. Chapter 3 is motivated by instances of Example 1 that arise in the analysis of single cell RNA sequencing data. In this setting, the typical assumption of Gaussianity is inappropriate, and the hypothesis generation procedures tend to be too complex to characterize the selection event analytically. The hypothesis generation procedures involve many steps, from preprocessing to variable selection to UMAP visualization to clustering. Different biologists may make slightly different choices at any given step of this preprocessing pipeline, and it is impractical to develop a new selective inference framework for every possible combination of choices.

In this complex setting, an appealing alternative is *sample splitting* [Cox, 1975]. We split the observations in our dataset into a training set and a test set. We can then generate a hypothesis based on the training set and test it on the test set (Example 1), or fit a model

to the training set and evaluate it on the test set (Example 2). Crucially, this procedure is a flexible wrapper; it does not need to be tailored to individual hypothesis generation procedures or specialized evaluation procedures. Unfortunately, traditional sample splitting is not always an option in unsupervised scenarios related to Example 1 [Neufeld et al., 2022a, Gao et al., 2022, Chen and Witten, 2023, Chen et al., 2021] and Example 2 [Bro et al., 2008, Owen and Perry, 2009, Fu and Perry, 2020, Tibshirani and Walther, 2005]. In the analysis of single cell RNA sequencing data, scientists frequently cluster the cells to estimate membership to discrete cell types, and then wish to test to see if a certain gene has different average expression across two cell types. This involves testing a hypothesis about average gene expression that was generated using the data (Example 1), and is the motivating problem for Chapter 3. This is a setting in which traditional sample splitting is not an option (see Gao et al. [2022] or Chen and Witten [2023] for more information).

To circumvent this problem, in Chapter 3, we develop *Poisson count splitting,* which is a flexible alternative to sample splitting that allows us to avoid double dipping in this unsupervised setting. Given a dataset X with n observations and p features, count splitting returns a training set and a test set that each also have dimension $n \times p$. Rather than splitting the observations, we split each individual count X_{ij} into X_{ij}^{train} and X_{ij}^{test} using binomial sampling. A classical result tells us that, if $X_{ij} \sim \text{Poisson}(\lambda_{ij})$, then X_{ij}^{train} and X_{ij}^{test} are independent and each follow Poisson distributions, where the parameter Λ_{ij} has been scaled by a known constant for each. This classical result has been used in other recent papers to avoid double dipping in settings related to Example 1 and Example 2 in both supervised and unsupervised settings [Chen et al., 2021, Leiner et al., 2022, Gerard, 2020, Sarkar and Stephens, 2021, Oliveira et al., 2022]. After count splitting, a hypothesis can be generated using X^{train} and tested using X^{test} (Example 1), or a model can be fit using X^{train} and evaluated using X^{test} (Example 2). In Chapter 3, we show that Poisson count splitting is useful for the task of testing for differential expression of genes across estimated latent variables (Example 1).

Single-cell RNA sequencing is often thought to be overdispersed relative to the Poisson

distribution [Choudhary and Satija, 2022], rendering a negative binomial assumption more appropriate. If Poisson count splitting is applied to an observation X_{ij} that actually follows a negative binomial distribution, there is positive correlation between X_{ij}^{train} and X_{ij}^{test} that increases as the amount of overdispersion increases. If Poisson count splitting is applied in such a setting, then the Type 1 error rate will not be controlled (Example 1), and model evaluation techniques will favor models that overfit the training data (Example 2). In Chapter 4, we extend Poisson count splitting to introduce *negative binomial count splitting*, which uses a new result to split a single count X_{ij} into independent training and test components X_{ij}^{train} and X_{ij}^{test} under the less restrictive and more general negative binomial assumption. We also show that count splitting is useful in determining how many cell types to retain in an analysis. By extending Poisson and negative binomial count splitting to create multiple independent folds of data, we can develop a natural alternative to cross validation via sample splitting that is useful for model evaluation and selection (Example 2).

Both Poisson and negative binomial count splitting are useful in contexts beyond the analysis of single cell RNA sequencing data. They can be used for inference after hypothesis generation or model evaluation in both supervised and unsupervised settings. Recognizing this broad utility of the count splitting approach, in Chapter 5 we extend the count splitting framework to a wide variety of distributions, and we name this generalized framework *data thinning*. Specifically, whenever X_{ij} comes from a convolution-closed distribution (a wide class of distributions that includes the Gaussian, gamma, negative binomial, Poisson, and binomial distributions), we provide a recipe for splitting X_{ij} into M mutually independent folds of data $\left(X_{ij}^{(1)}, \ldots, X_{ij}^{(M)}\right)$ such that $X_{ij} = \sum_{m=1}^{M} X_{ij}^{(m)}$, and each component follows the same distribution as X, up to a known scaling of a parameter. This procedure leads to an alternative to cross validation via sample splitting that can be applied in both supervised and unsupervised settings.

We are not the first to propose an alternative to sample splitting that splits a single realization of data (rather than splitting the observations). Inspired by Tian and Taylor [2018]'s use of randomized responses, Rasines and Young [2022] define what they call the

(U, V)-decomposition, which injects independent noise W into a dataset X to create two independent random variables $U = u(X, W)$ and $V = v(X, W)$. However, they do not describe how to perform a (U, V)-decomposition other than in the special case of a Gaussian random vector with known covariance. Our data thinning framework achieves the goal set out in their paper, providing a concrete recipe for finding such decompositions in a broad set of examples. Another paper with a similar goal is Leiner et al. [2022]. They define "data fission", which seeks to find random variables $f(X)$ and $g(X)$ for which the distributions of $f(X)$ and $g(X) \mid f(X)$ are known and for which $X = h(f(X), g(X))$. When these two random variables are independent (which they describe as the "P1" property), their proposal aligns with the goals of data thinning. However, like Rasines and Young [2022], they do not provide a general strategy for performing P1-fission, and the only two examples they provide are the Gaussian vector with known covariance and the Poisson: two decompositions which were previously available in the literature.

In Chapter 6, we review the overall advantages and disadvantages of selective inference, sample splitting, and data thinning in the context of Examples 1 and 2, and then discuss ongoing work to extend and improve data thinning.

Chapter 2

SELECTIVE INFERENCE FOR REGRESSION TREES

The contents of this chapter are published in the Journal of Machine Learning Research [Neufeld et al., 2022b].

2.1 Introduction

Regression tree algorithms recursively partition covariate space using binary splits to obtain regions that are maximally homogeneous with respect to a continuous response. The Classification and Regression Tree (CART; Breiman et al. 1984) proposal, which involves growing a large tree and then pruning it back, is by far the most popular of these algorithms.

The regions defined by the splits in a fitted CART tree induce a piecewise constant regression model where the predicted response within each region is the mean of the observations in that region. CART is popular in large part because it is highly interpretable; someone without technical expertise can easily read the tree to make predictions, and to understand why a certain prediction is made. However, its interpretability belies the fact that CART trees are highly unstable: a small change to the training dataset can drastically change the structure of the fitted tree. In the absence of an established notion of statistical significance associated with a given split in the tree, it is hard for a practitioner to know whether they are interpreting signal or noise. In this chapter, we use the framework of selective inference to fill this gap by providing a toolkit to conduct inference on hypotheses motivated by the output of the CART algorithm.

Given a CART tree, consider testing for a difference in the mean response of the regions resulting from a binary split. A very naive approach, such as a two-sample Z-test, that does not account for the fact that the regions were themselves estimated from the data will fail

to control the selective Type 1 error rate: the probability of rejecting a true null hypothesis, given that we decided to test it [Fithian et al., 2014]. Similarly, a naive Z-interval for the mean response in a region will not attain nominal selective coverage: the probability that the interval covers the parameter, given that we chose to construct it.

In fact, approaches for conducting inference on the output of a regression tree are quite limited. Sample splitting involves fitting a CART tree using a subset of the observations, which will naturally lead to an inferior tree to the one resulting from all of the observations, and thus is unsatisfactory in many applied settings; see Athey and Imbens [2016]. Wager and Walther [2015] develop convergence guarantees for unpruned CART trees that can be leveraged to build confidence intervals for the mean response within a region; however, they do not provide finite-sample results and cannot accommodate pruning. Loh et al. [2016] and Loh et al. [2019] develop bootstrap calibration procedures that attempt to provide confidence intervals for the regions of a regression tree. In Appendix A.1, we show that this bootstrap calibration approach fails to provide intervals that achieve nominal coverage for the parameters of interest in this chapter.

As an alternative to performing inference on a CART tree, one could turn to the conditional inference tree (CTree) framework of Hothorn et al. [2006]. This framework uses a different tree-growing algorithm than CART, and at each split tests for linear association between the split covariate and the response. As summarized in Loh [2014], the CTree framework alleviates issues with instability and variable selection bias associated with CART. Despite these advantages, CTree remains far less widely-used than CART. Furthermore, while CTree attaches a notion of statistical significance to each split in a tree, it does not directly allow for inference on the mean response within a region or the difference in mean response between two regions. Finally, while the CTree framework requires few assumptions, its inference is based on asymptotics.

In this chapter, we introduce a finite-sample selective inference [Fithian et al., 2014] framework for the difference between the mean responses in two regions, and for the mean response in a single region, in a pruned or unpruned CART tree. We condition on the event

that CART yields a particular set of regions, and thereby achieve selective Type 1 error rate control as well as nominal selective coverage.

The rest of this chapter is organized as follows. In Section 2.2, we review the CART algorithm, and briefly define some key ideas in selective inference. In Section 2.3, we present our proposal for selective inference on the regions estimated via CART. We show that the necessary conditioning sets can be efficiently computed in Section 2.4. In Section 2.5 we compare our framework to sample splitting and CTree via simulation. In Section 2.6 we compare our framework to CTree on data from the Box Lunch Study. The discussion is in Section 2.7. Technical details are relegated to the supplementary materials.

2.2 Background

2.2.1 Notation for regression trees

Given p covariates $(X_1, \ldots, X_p)$ measured on each of n observations $(x_1, \ldots, x_n)$, let $x_{j,(s)}$ denote the sth order statistic of the jth covariate, and define the half-spaces

$$\chi_{j,s,1} = \left\{ z \in \mathbb{R}^p : z_j \leq x_{j,(s)} \right\}, \qquad \chi_{j,s,0} = \left\{ z \in \mathbb{R}^p : z_j > x_{j,(s)} \right\}. \tag{2.1}$$

The following definitions are illustrated in Figure 2.1.

Definition 2.1 (Tree and Region) *Consider a set $\mathcal{S}$ such that $R \subseteq \mathbb{R}^p$ for all $R \in \mathcal{S}$. Then $\mathcal{S}$ is a* tree *if and only if (i) $\mathbb{R}^p \in \mathcal{S}$; (ii) every element of $\mathcal{S} \setminus \mathbb{R}^p$ equals $R \cap \chi_{j,s,e}$ for some $R \in \mathcal{S}$, $j \in \{1, \ldots, p\}$, $s \in \{1, \ldots, n-1\}$, $e \in \{0, 1\}$; (iii) $R \cap \chi_{j,s,e} \in \mathcal{S}$ implies that $R \cap \chi_{j,s,1-e} \in \mathcal{S}$ for $e \in \{0, 1\}$; and (iv) for any $R, R' \in \mathcal{S}, R \cap R' \in \{\emptyset, R, R'\}$. If $R \in \mathcal{S}$ and $\mathcal{S}$ is a tree, then we refer to R as a* region.

We use the notation TREE to refer to a particular tree. Definition 2.1 implies that any region $R \in \text{TREE} \setminus \{\mathbb{R}^p\}$ is of the form $R = \cap_{l=1}^{L} \chi_{j_l,s_l,e_l}$, where for each $l = 1, \ldots, L$, we have that $j_l \in \{1, \ldots, p\}$, $s_l \in \{1, \ldots, n-1\}$, and $e_l \in \{0, 1\}$. We call L the *level* of the region, and use the convention that the level of $\mathbb{R}^p$ is 0.

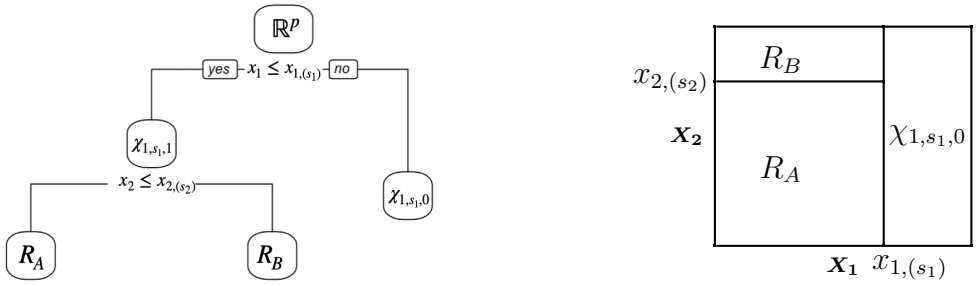


Figure 2.1: The regression tree takes the form $\text{TREE} = \{\mathbb{R}^p, \chi_{1,s_1,1}, \chi_{1,s_1,0}, \chi_{1,s_1,1} \cap \chi_{2,s_2,1}, \chi_{1,s_1,1} \cap \chi_{2,s_2,0}\}$. The regions $R_A = \chi_{1,s_1,1} \cap \chi_{2,s_2,1}$ and $R_B = \chi_{1,s_1,1} \cap \chi_{2,s_2,0}$ are siblings, and are children, and therefore descendants, of the region $\chi_{1,s_1,1}$. The ancestors of R_A and R_B are $\mathbb{R}^p$ and $\chi_{1,s_1,1}$. Furthermore, R_A, R_B, and $\chi_{1,s_1,0}$ are terminal regions.

Definition 2.2 (Siblings and Children) *Suppose that* $\{R, R \cap \chi_{j,s,1}, R \cap \chi_{j,s,0}\} \subseteq \text{TREE}$. *Then* $R \cap \chi_{j,s,1}$ *and* $R \cap \chi_{j,s,0}$ *are* siblings. *Furthermore, they are the* children *of* R.

Definition 2.3 (Descendant and Ancestor) *If* $R, R' \in \text{TREE}$ *and* $R \subseteq R'$, *then* R *is a* descendant *of* R', *and* R' *is an* ancestor *of* R.

Definition 2.4 (Terminal Region) *A region* $R \in \text{TREE}$ *without descendants is a* terminal region.

We let $\text{DESC}(R, \text{TREE})$ denote the set of descendants of region R in TREE, and we let $\text{TERM}(R, \text{TREE})$ denote the subset of $\text{DESC}(R, \text{TREE})$ that are terminal regions.

Given a response vector $y \in \mathbb{R}^n$, let $\bar{y}_R = \left(\sum_{i:x_i \in R} y_i\right) / \left\{\sum_{i=1}^n 1_{(x_i \in R)}\right\}$, where $1_{(A)}$ is an indicator variable that equals 1 if the event A holds, and 0 otherwise. Then, a tree TREE induces the regression model $\hat{\mu}(x) = \sum_{R \in \text{TERM}(\mathbb{R}^p, \text{TREE})} \bar{y}_R 1_{(x \in R)}$. In other words, it predicts the response within each terminal region to be the mean of the observations in that region.

2.2.2 A review of the CART algorithm [Breiman et al., 1984]

The CART algorithm [Breiman et al., 1984] greedily searches for a tree that minimizes the sum of squared errors $\sum_{R \in \text{TERM}(\mathbb{R}^p, \text{TREE})} \sum_{i:x_i \in R} (y_i - \bar{y}_R)^2$. It first grows a very large tree via

recursive binary splits, starting with the full covariate space $\mathbb{R}^p$. To split a region R, it selects the covariate x_j and the split point $x_{j,(s)}$ to maximize the *gain*, defined as

$$\textsc{gain}_R(y, j, s) \equiv \sum_{i \in R} (y_i - \bar{y}_R)^2 - \left\{ \sum_{i \in R \cap \chi_{j,s,1}} \left(y_i - \bar{y}_{R \cap \chi_{j,s,1}}\right)^2 + \sum_{i \in R \cap \chi_{j,s,0}} \left(y_i - \bar{y}_{R \cap \chi_{j,s,0}}\right)^2 \right\}. \quad (2.2)$$

Details are provided in Algorithm 2.1.

Once a very large tree has been grown, cost-complexity pruning is applied. We define the average per-region gain in sum-of-squared errors provided by the descendants of a region R,

$$g(R, \textsc{tree}, y) = \frac{\sum_{i:x_i \in R} (y_i - \bar{y}_R)^2 - \sum_{r \in \textsc{term}(R, \textsc{tree})} \sum_{i:x_i \in r} (y_i - \bar{y}_r)^2}{|\,\textsc{term}(R, \textsc{tree})| - 1}. \quad (2.3)$$

Given a complexity parameter $\lambda \geq 0$, if $g(R, \textsc{tree}, y) < \lambda$ for some $R \in \textsc{tree}$, then cost-complexity pruning removes R's descendants from $\textsc{tree}$, turning R into a terminal region. Details are in Algorithm 2.2, which involves the notion of a *bottom-up ordering*.

Definition 2.5 (Bottom-up ordering) *Let* $\textsc{tree} = \{R_1, \ldots, R_K\}$. *Let* π *be a permutation of the integers* $(1, \ldots, K)$. *Then* $\mathcal{O} = \left(R_{\pi(1)}, \ldots, R_{\pi(K)}\right)$ *is a* bottom-up ordering *of the regions in* $\textsc{tree}$ *if, for all* $k = 1, \ldots, K$, $\pi(k) \leq \pi(j)$ *if* $R_k \in \textsc{desc}(R_j, \textsc{tree})$.

There are other equivalent formulations for cost-complexity pruning (see Proposition 7.2 in Ripley [1996]); the formulation in Algorithm 2.2 is convenient for establishing the results in this chapter.

To summarize, the CART algorithm first applies Algorithm 2.1 to the initial region $\mathbb{R}^p$ and the data y to obtain an unpruned tree, which we call $\textsc{tree}^0(y)$. It then applies Algorithm 2.2 to $\textsc{tree}^0(y)$ to obtain an optimally-pruned tree using complexity parameter λ, which we call $\textsc{tree}^\lambda(y)$.

Algorithm 2.1 (Growing a tree)

GROW*(R, y)*

1. *If a stopping condition is met, return R.*

2. *Else return* $\{R$, GROW$(R \cap \chi_{\tilde{j},\tilde{s},1}, y)$, GROW$(R \cap \chi_{\tilde{j},\tilde{s},0}, y)\}$, *where*
$$(\tilde{j}, \tilde{s}) \in \arg\max_{(j,s):s\in\{1,\ldots,n-1\},j\in\{1,\ldots,p\}} \text{GAIN}_R\,(y, j, s)\,.$$

Algorithm 2.2 (Cost-complexity pruning) *Parameter $\mathcal{O}$ is a bottom-up ordering of the K regions in* TREE.
PRUNE(TREE, y, λ, $\mathcal{O}$)

1. *Let* $\text{TREE}_0 = \text{TREE}$. *Let K be the number of regions in* TREE_0.
2. *For* $k = 1, \ldots, K$:
   (a) *Let R be the kth region in $\mathcal{O}$.*
   (b) *Update* TREE_k *as follows, where $g(\cdot)$ is defined in* (2.3):
$$\text{TREE}_k \leftarrow \begin{cases} \text{TREE}_{k-1} \setminus \text{DESC}(R, \text{TREE}_{k-1}) & \text{if } g(R, \text{TREE}_{k-1}, y) < \lambda, \\ \text{TREE}_{k-1} & \text{otherwise.} \end{cases}$$
3. *Return* TREE_K.

2.2.3 A brief overview of selective inference

Here, we provide a very brief overview of selective inference; see Fithian et al. [2014] or Taylor and Tibshirani [2015] for a more detailed treatment.

Consider conducting inference on a parameter θ. Classical approaches assume that we were already interested in conducting inference on θ before looking at our data. If, instead, our interest in θ was sparked by looking at our data, then inference must be performed with care: we must account for the fact that we "selected" θ based on the data [Fithian et al., 2014]. In this setting, interest focuses on a p-value $p(Y)$ such that the test for $H_0 : \theta = \theta_0$ based on $p(Y)$ controls the *selective Type 1 error* rate, in the sense that

$$pr_{H_0:\theta=\theta_0}\{p(Y) \leq \alpha \mid \theta \text{ selected}\} \leq \alpha, \text{ for all } 0 \leq \alpha \leq 1. \tag{2.4}$$

Also of interest are confidence intervals $[L(Y), U(Y)]$ that achieve $(1-\alpha)$-*selective coverage*

for the parameter θ, meaning that

$$pr\left\{\theta \in [L(Y), U(Y)] \mid \theta \text{ selected}\right\} \geq 1 - \alpha. \tag{2.5}$$

Roughly speaking, the inferential guarantees in (2.4) and (2.5) can be achieved by defining p-values and confidence intervals that condition on the aspect of the data that led to the selection of θ. In recent years, a number of papers have taken this approach to perform selective inference on parameters selected from the data in the regression [Lee et al., 2016, Liu et al., 2018, Tian and Taylor, 2018, Tibshirani et al., 2016], clustering [Gao et al., 2022], and changepoint detection [Hyun et al., 2021, Jewell et al., 2022] settings.

In the next section, we propose p-values that satisfy (2.4) and confidence intervals that satisfy (2.5) in the setting of CART, where the parameter of interest is either the mean response within a region, or the difference between the mean responses of two sibling regions.

2.3 The selective inference framework for CART

2.3.1 Inference on a pair of sibling regions

Throughout this chapter, we assume that $Y \sim N_n(\mu, \sigma^2 I_n)$ with $\sigma > 0$ known.

We let $X \in \mathbb{R}^{n \times p}$ denote a fixed covariate matrix. Suppose that we apply CART with complexity parameter λ to a realization $y = (y_1, \ldots, y_n)^{\mathrm{T}}$ from Y to obtain $\textsc{tree}^{\lambda}(y)$. Given sibling regions R_A and R_B in $\textsc{tree}^{\lambda}(y)$, we define a contrast vector $\nu_{sib} \in \mathbb{R}^n$ such that

$$(\nu_{sib})_i = \frac{1_{(x_i \in R_A)}}{\sum_{i'=1}^n 1_{(x_{i'} \in R_A)}} - \frac{1_{(x_i \in R_B)}}{\sum_{i'=1}^n 1_{(x_{i'} \in R_B)}}, \tag{2.6}$$

and $\nu_{sib}^{\mathrm{T}}\mu = \left(\sum_{i:x_i \in R_A} \mu_i\right) / \left\{\sum_{i=1}^n 1_{(x_i \in R_A)}\right\} - \left(\sum_{i:x_i \in R_B} \mu_i\right) / \left\{\sum_{i=1}^n 1_{(x_i \in R_B)}\right\}$. Now, consider testing the null hypothesis of no difference in means between R_A and R_B, i.e. $H_0 : \nu_{sib}^{\mathrm{T}}\mu = 0$ versus $H_1 : \nu_{sib}^{\mathrm{T}}\mu \neq 0$. This null hypothesis is of interest because R_A and R_B appeared as siblings in $\textsc{tree}^{\lambda}(y)$. A test based on a p-value of the form $pr_{H_0}\left(|\nu_{sib}^{\mathrm{T}}Y| \geq |\nu_{sib}^{\mathrm{T}}y|\right)$ that does not account for this will not control the selective Type 1 error rate in (2.4).

To control the selective Type 1 error rate, we propose a p-value that conditions on the aspect of the data that led us to select $\nu_{sib}^{\mathrm{T}}\mu$,

$$pr_{H_0}\left\{|\nu_{sib}^{\mathrm{T}}Y| \geq |\nu_{sib}^{\mathrm{T}}y| \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\right\}. \tag{2.7}$$

But (2.7) depends on a nuisance parameter, the portion of μ that is orthogonal to ν_{sib}. To remove the dependence on this nuisance parameter, we condition on its sufficient statistic $\mathcal{P}^{\perp}_{\nu_{sib}}Y$, where $\mathcal{P}^{\perp}_{\nu} = I - \nu\nu^{\mathrm{T}}/\|\nu\|_2^2$. The resulting p-value, or "tree-value", is defined as

$$p_{sib}(y) = pr_{H_0}\left\{|\nu_{sib}^{\mathrm{T}}Y| \geq |\nu_{sib}^{\mathrm{T}}y| \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y), \mathcal{P}^{\perp}_{\nu_{sib}}Y = \mathcal{P}^{\perp}_{\nu_{sib}}y\right\}. \tag{2.8}$$

Results similar to Theorem 2.1 can be found in Jewell et al. [2022], Lee et al. [2016], Liu et al. [2018], and Tibshirani et al. [2016].

Theorem 2.1 *The test based on the p-value $p_{sib}(y)$ in (2.8) controls the selective Type 1 error rate for $H_0 : \nu_{sib}^{\mathrm{T}}\mu = 0$, where ν_{sib} is defined in (2.6), in the sense that*

$$pr_{H_0}\left\{p_{sib}(Y) \leq \alpha \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\right\} = \alpha, \text{ for all } 0 \leq \alpha \leq 1. \tag{2.9}$$

Furthermore, $p_{sib}(y) = pr\left\{|\phi| \geq |\nu_{sib}^{\mathrm{T}}y| \mid \phi \in S^{\lambda}_{sib}(\nu_{sib})\right\}$, where $\phi \sim N(0, \|\nu_{sib}\|_2^2\sigma^2)$, $y'(\phi, \nu) = \mathcal{P}^{\perp}_{\nu}y + \phi(\nu/\|\nu\|_2^2)$, and

$$S^{\lambda}_{sib}(\nu_{sib}) = \{\phi : R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}\}. \tag{2.10}$$

Proofs of all theoretical results are provided in the appendix. Theorem 2.1 says that given the set $S^{\lambda}_{sib}(\nu_{sib})$, we can compute the p-value in (2.8) using

$$p_{sib}(y) = 1 - F\left\{|\nu_{sib}^{\mathrm{T}}y|; 0, \|\nu_{sib}\|_2^2\sigma^2, S^{\lambda}_{sib}(\nu_{sib})\right\} + F\left\{-|\nu_{sib}^{\mathrm{T}}y|; 0, \|\nu_{sib}\|_2^2\sigma^2, S^{\lambda}_{sib}(\nu_{sib})\right\}, \tag{2.11}$$

where $F(\ \cdot\ ; 0, \|\nu\|^2\sigma^2, S)$ denotes the cumulative distribution function of the $N(0, \|\nu\|_2^2\sigma^2)$

distribution truncated to the set S. In Section 2.4, we provide an efficient approach for analytically characterizing the truncation set $S^{\lambda}_{sib}(\nu_{sib})$. To avoid numerical issues associated with the truncated normal distribution, we compute (2.11) using methods described in the supplement of Chen and Bien [2020]. Note that the proof of Theorem 2.1, and consequently the efficient computation of $p_{sib}(y)$ discussed in Section 2.4, relies on the assumption that $Y \sim N_n(\mu, \sigma^2 I_n)$.

We now consider inverting the test proposed in (2.8) to construct an equitailed confidence interval for $\nu_{sib}^{\mathrm{T}}\mu$ that has $(1-\alpha)$-selective coverage (2.5), in the sense that

$$pr\left\{\nu_{sib}^{\mathrm{T}}\mu \in [L(Y), U(Y)] \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\right\} = 1-\alpha. \tag{2.12}$$

Proposition 2.3.1 *For any $0 \leq \alpha \leq 1$ and any realization $y \in \mathbb{R}^n$, the values $L(y)$ and $U(y)$ that satisfy*

$$F\{\nu_{sib}^{\mathrm{T}}y; L(y), \sigma^2\|\nu_{sib}\|_2^2, S^{\lambda}_{sib}(\nu_{sib})\} = 1-\alpha/2, \quad F\{\nu_{sib}^{\mathrm{T}}y; U(y), \sigma^2\|\nu_{sib}\|_2^2, S^{\lambda}_{sib}(\nu_{sib})\} = \alpha/2, \tag{2.13}$$

are unique, and $[L(Y), U(Y)]$ achieves $(1-\alpha)$-selective coverage for $\nu_{sib}^{\mathrm{T}}\mu$.

2.3.2 Inference on a single region

Given a single region R_A in a CART tree, we define the contrast vector ν_{reg} such that

$$(\nu_{reg})_i = 1_{(x_i \in R_A)} \Big/ \left\{\sum_{i'=1}^{n} 1_{(x_{i'} \in R_A)}\right\}. \tag{2.14}$$

Then, $\nu_{reg}^{\mathrm{T}}\mu = \left(\sum_{i:x_i \in R_A} \mu_i\right) / \left\{\sum_{i=1}^{n} 1_{(x_i \in R_A)}\right\}$. We now consider testing the null hypothesis $H_0 : \nu_{reg}^{\mathrm{T}}\mu = c$ for some fixed c. Because our interest in this null hypothesis results from the fact that $R_A \in \textsc{tree}^{\lambda}(y)$, we must condition on this event in defining the p-value. We define

$$p_{reg}(y) = pr_{H_0}\left\{|\nu_{reg}^{\mathrm{T}}Y - c| \geq |\nu_{reg}^{\mathrm{T}}y - c| \mid R_A \in \textsc{tree}^{\lambda}(Y), \mathcal{P}^{\perp}_{\nu_{reg}}Y = \mathcal{P}^{\perp}_{\nu_{reg}}y\right\}, \tag{2.15}$$

and introduce the following theorem.

Theorem 2.2 *The test based on the p-value $p_{reg}(y)$ in (2.15) controls the selective Type 1 error rate for $H_0 : \nu_{reg}^{\mathrm{T}}\mu = c$, where ν_{reg} is defined in (2.14). Furthermore, $p_{reg}(y) = pr\left\{|\phi - c| \geq |\nu_{reg}^{\mathrm{T}}y - c| \mid \phi \in S_{reg}(\nu_{reg})\right\}$, where $\phi \sim N(c, \|\nu_{reg}\|_2^2\sigma^2)$ and, for $y'(\phi, \nu) = \mathcal{P}_\nu^\perp y + \phi(\nu/\|\nu\|_2^2)$,*

$$S_{reg}^{\lambda}(\nu_{reg}) = \{\phi : R_A \in \textsc{tree}^{\lambda}\{y'(\phi, \nu_{reg})\}\}. \tag{2.16}$$

Theorem 2 and the resulting efficient computations in Section 2.4 rely on the assumption that $Y \sim N_n(\mu, \sigma^2 I_n)$.

We can also define a confidence interval for $\nu_{reg}^{\mathrm{T}}\mu$ that attains nominal selective coverage.

Proposition 2.3.2 *For any $0 \leq \alpha \leq 1$ and any realization $y \in \mathbb{R}^n$, the values $L(y)$ and $U(y)$ that satisfy*

$$F\{\nu_{reg}^{\mathrm{T}}y; L(y), \sigma^2\|\nu_{reg}\|_2^2, S_{reg}^{\lambda}(\nu_{reg})\} = 1 - \alpha/2, \quad F\{\nu_{reg}^{\mathrm{T}}y; U(y), \sigma^2\|\nu_{reg}\|_2^2, S_{reg}^{\lambda}(\nu_{reg})\} = \alpha/2, \tag{2.17}$$

are unique, and $[L(Y), U(Y)]$ achieves $(1-\alpha)$-selective coverage for $\nu_{reg}^{\mathrm{T}}\mu$.

In Section 2.4, we propose an approach to analytically characterize the set $S_{reg}^{\lambda}(\nu_{reg})$ in (2.16).

2.3.3 Intuition for the conditioning sets $S_{sib}^{\lambda}(\nu_{sib})$ and $S_{reg}^{\lambda}(\nu_{reg})$

We first develop intuition for the set $S_{sib}^{\lambda}(\nu_{sib})$ defined in (2.10). From Theorem 2.1,

$$\{y'(\phi, \nu_{sib})\}_i = y_i + (\phi - \nu_{sib}^{\mathrm{T}}y)\left\{\frac{\sum_{i'=1}^n 1_{(x_{i'} \in R_B)}}{\sum_{i'=1}^n 1_{(x_{i'} \in R_A \cup R_B)}} 1_{(x_i \in R_A)} - \frac{\sum_{i'=1}^n 1_{(x_{i'} \in R_A)}}{\sum_{i'=1}^n 1_{(x_{i'} \in R_A \cup R_B)}} 1_{(x_i \in R_B)}\right\}.$$

Thus, $y'(\phi, \nu_{sib})$ is a perturbation of y that exaggerates the difference between the observed sample mean responses of R_A and R_B if $|\phi| > |\nu_{sib}^{\mathrm{T}}y|$, and shrinks that difference if $|\phi| < |\nu_{sib}^{\mathrm{T}}y|$. The set $S_{sib}^{\lambda}(\nu_{sib})$ quantifies the amount that we can shift the difference in sample mean responses between R_A and R_B while still producing a tree containing these sibling

regions. The top row of Figure 2.2 displays $\text{TREE}^0\{y'(\phi, \nu_{sib})\}$, as a function of ϕ, in an example where $S^0_{sib}(\nu_{sib}) = (-19.8, -1.8) \cup (0.9, 34.9)$.

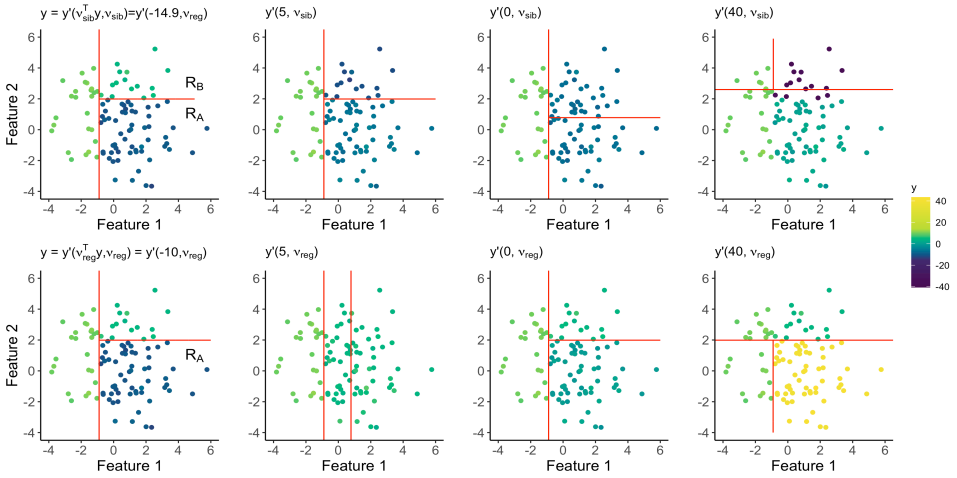


Figure 2.2: Data with $n = 100$ and $p = 2$. Regions resulting from CART ($\lambda = 0$) are delineated using solid lines. Here, $R_A = \chi_{1,26,0} \cap \chi_{2,72,1}$ and $R_B = \chi_{1,26,0} \cap \chi_{2,72,0}$. *Top:* Output of CART applied to $y'(\phi, \nu_{sib})$, where ν_{sib} in (2.6) encodes the contrast between R_A and R_B, for various values of ϕ. The left-most panel displays $y = y'(\nu^{\text{T}}_{sib} y, \nu_{sib})$. By inspection, we see that $-14.9 \in S^0_{sib}(\nu_{sib})$ and $5 \in S^0_{sib}(\nu_{sib})$, but $0 \notin S^0_{sib}(\nu_{sib})$ and $40 \notin S^0_{sib}(\nu_{sib})$. In fact, $S^0_{sib}(\nu_{sib}) = (-19.8, -1.8) \cup (0.9, 34.9)$. *Bottom:* Output of CART applied to $y'(\phi, \nu_{reg})$, where ν_{reg} in (2.14) encodes membership in R_A. The left-most panel displays $y = y'(\nu^{\text{T}}_{reg} y, \nu_{reg})$. Here, $S^0_{reg}(\nu_{reg}) = (-\infty, 3.1) \cup (5.8, 8.8) \cup (14.1, \infty)$.

We next develop intuition for $S^\lambda_{reg}(\nu_{reg})$, defined in (2.16). Note that $\{y'(\phi, \nu_{reg})\}_i = y_i + (\phi - \nu^{\text{T}}_{reg} y) 1_{(x_i \in R_A)}$, where $y'(\phi, \nu_{reg})$ is defined in Theorem 2.2. Thus, $y'(\phi, \nu_{reg})$ shifts the responses of the observations in R_A so that their sample mean equals ϕ, and leaves the others unchanged. The set $S^\lambda_{reg}(\nu_{reg})$ quantifies the amount that we can exaggerate or shrink the sample mean response in region R_A while still producing a tree that contains R_A. The bottom row of Figure 2.2 displays $y'(\phi, \nu_{reg})$ as ϕ is varied, in an example with $S^0_{reg}(\nu_{reg}) = (-\infty, 3.1) \cup (5.8, 8.8) \cup (14.1, \infty)$.

2.4 Computing the conditioning sets $S^{\lambda}_{sib}(\nu_{sib})$ and $S^{\lambda}_{reg}(\nu_{reg})$

2.4.1 Recharacterizing the conditioning sets in terms of branches

We begin by introducing the concept of a branch.

Definition 2.6 (Branch) *A* branch *is an ordered sequence of triples* $\mathcal{B} = ((j_1, s_1, e_1), \ldots, (j_L, s_L, e_L))$ *such that* $j_l \in \{1, \ldots, p\}$, $s_l \in \{1, \ldots, n-1\}$, *and* $e_l \in \{0, 1\}$ *for* $l = 1, \ldots, L$. *The branch* $\mathcal{B}$ *induces a nested set of regions* $\mathcal{R}(\mathcal{B}) = \{R^{(0)}, R^{(1)}, \ldots, R^{(L)}\}$, *where* $R^{(l)} = \bigcap_{l'=1}^{l} \chi_{j_{l'}, s_{l'}, e_{l'}}$ *for* $l = 1, \ldots, L$, *and* $R^{(0)} = \mathbb{R}^p$.

For a branch $\mathcal{B}$ and a vector ν, we define

$$S^{\lambda}(\mathcal{B}, \nu) = \left\{\phi : \mathcal{R}(\mathcal{B}) \subseteq \textsc{tree}^{\lambda}\{y'(\phi, \nu)\}\right\}. \tag{2.18}$$

For $R \in \textsc{tree}$, we let $\textsc{branch}(R, \textsc{tree})$ denote the branch such that $\mathcal{R}\{\textsc{branch}(R, \textsc{tree})\}$ contains R and all of its ancestors in $\textsc{tree}$.

Lemma 2.1 *Suppose that* R_A *and* R_B *are siblings in* $\textsc{tree}^{\lambda}(y)$. *Then* R_A *and* R_B *are siblings in* $\textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}$ *if and only if* $\mathcal{R}[\textsc{branch}\{R_A, \textsc{tree}^{\lambda}(y)\}] \subseteq \textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}$. *Therefore,* $S^{\lambda}_{sib}(\nu_{sib}) = S^{\lambda}[\textsc{branch}\{R_A, \textsc{tree}^{\lambda}(y)\}, \nu_{sib}]$, *defined in* (2.10) *and* (2.18).

Lemma 2.1 says that $\textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}$ contains siblings R_A and R_B if and only if it contains the entire branch associated with R_A in $\textsc{tree}^{\lambda}(y)$. However, Lemma 2.1 does not apply in the single region case: for ν_{reg} defined in (2.14) and some $R_A \in \textsc{tree}^{\lambda}(y)$, the fact that $R_A \in \textsc{tree}^{\lambda}\{y'(\phi, \nu_{reg})\}$ does not imply that $\mathcal{R}[\textsc{branch}\{R_A, \textsc{tree}^{\lambda}(y)\}] \subseteq \textsc{tree}^{\lambda}\{y'(\phi, \nu_{reg})\}$. Instead, a result similar to Lemma 2.1 holds, involving permutations of the branch.

Definition 2.7 (Permutation of a branch) *Let* Π *denote the set of all* $L!$ *permutations of* $(1, 2, \ldots, L)$. *Given* $\pi \in \Pi$ *and a branch* $\mathcal{B} = ((j_1, s_1, e_1), \ldots, (j_L, s_L, e_L))$, *we say that* $\pi(\mathcal{B}) = ((j_{\pi(1)}, s_{\pi(1)}, e_{\pi(1)}), \ldots, (j_{\pi(L)}, s_{\pi(L)}, e_{\pi(L)}))$ *is a* permutation of the branch $\mathcal{B}$.

Branch $\mathcal{B}$ and its permutation $\pi(\mathcal{B})$ induce the same region $R^{(L)}$, but $\mathcal{R}\{\pi(\mathcal{B})\} \neq \mathcal{R}(\mathcal{B})$.

Lemma 2.2 *Let $R_A \in \text{TREE}^\lambda(y)$. Then $R_A \in \text{TREE}^\lambda\{y'(\phi, \nu_{reg})\}$ if and only if there exists a $\pi \in \Pi$ such that $\mathcal{R}[\pi\{\text{BRANCH}_{R_A}(y)\}] \subseteq \text{TREE}^\lambda\{y'(\phi, \nu_{reg})\}$. Thus, for $S^\lambda_{reg}(\nu_{reg})$ in (2.16),*

$$S^\lambda_{reg}(\nu_{reg}) = \bigcup_{\pi \in \Pi} S^\lambda\left(\pi\left[\text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}\right], \nu_{reg}\right). \tag{2.19}$$

Lemmas 2.1 and 2.2 reveal that computing $S^\lambda_{sib}(\nu_{sib})$ and $S^\lambda_{reg}(\nu_{reg})$ requires characterizing sets of the form $S^\lambda(\mathcal{B}, \nu)$, defined in (2.18). To compute $S^\lambda_{sib}(\nu_{sib})$ we will only need to consider $S^\lambda(\mathcal{B}, \nu)$ where $\mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^\lambda(y)$. However, to compute $S^\lambda_{reg}(\nu_{reg})$, we will need to consider $S^\lambda\{\pi(\mathcal{B}), \nu\}$ where $\mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^\lambda(y)$ but $\mathcal{R}\{\pi(\mathcal{B})\} \nsubseteq \text{TREE}^\lambda(y)$.

2.4.2 Computing $S^\lambda(\mathcal{B}, \nu)$ in (2.18)

Throughout this section, we consider a vector $\nu \in \mathbb{R}^n$ and a branch $\mathcal{B} = ((j_1, s_1, e_1), \ldots, (j_L, s_L, e_L))$, where $\mathcal{R}(\mathcal{B})$ may or may not be in $\text{TREE}^\lambda(y)$. Recall from Definition 2.6 that $\mathcal{B}$ induces the nested regions $R^{(l)} = \bigcap_{l'=1}^{l} \chi_{j_{l'}, s_{l'}, e_{l'}}$ for $l = 1, \ldots, L$, and $R^{(0)} = \mathbb{R}^p$. Throughout this section, our only requirement on $\mathcal{B}$ and ν is the following condition.

Condition 2.1 *For $y'(\phi, \nu)$ defined in Theorem 2.1, $\mathcal{B}$ and ν satisfy $\{y'(\phi, \nu)\}_i = y_i + c_1 1_{\{x_i \in R^{(L)}\}} + c_2 1_{[x_i \in \{R^{(L-1)} \cap \chi_{j_L, s_L, 1-e_L}\}]}$ for $i = 1, \ldots, n$ and for some constants c_1 and c_2.*

To characterize $S^\lambda(\mathcal{B}, \nu)$ in (2.18), recall that the CART algorithm in Section 2.2.2 involves growing a very large tree $\text{TREE}^0(y)$, and then pruning it. We first characterize the set

$$S_{grow}(\mathcal{B}, \nu) = \{\phi : \mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^0\{y'(\phi, \nu)\}\}. \tag{2.20}$$

Proposition 2.4.1 *Recall the definition of $\text{GAIN}_{R^{(l)}}\{y'(\phi, \nu), j, s\}$ in (2.2), and let $S_{l,j,s} = \{\phi : \text{GAIN}_{R^{(l-1)}}\{y'(\phi, \nu), j, s\} \leq \text{GAIN}_{R^{(l-1)}}\{y'(\phi, \nu), j_l, s_l\}\}$. Then, $S_{grow}(\mathcal{B}, \nu) = \bigcap_{l=1}^{L} \bigcap_{j=1}^{p} \bigcap_{s=1}^{n-1} S_{l,j,s}$.*

Proposition 2.4.2 says that we can compute $S_{grow}(\mathcal{B}, \nu)$ efficiently.

Proposition 2.4.2 *The set $S_{l,j,s}$ is defined by a quadratic inequality in ϕ. Furthermore, we can evaluate all of the sets $S_{l,j,s}$, for $l = 1, \ldots, L$, $j = 1, \ldots, p$, $s = 1, \ldots, n-1$, in $O\{npL + np\log(n)\}$ operations. Intersecting these sets to obtain $S_{grow}(\mathcal{B}, \nu)$ requires at most $O\{npL \times \log(npL)\}$ operations, and only $O(npL)$ operations if $\mathcal{B} = \text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}$ and ν is of the form ν_{sib} in* (2.6).

Noting that $S^\lambda(\mathcal{B}, \nu) = \left\{\phi \in S_{grow}(\mathcal{B}, \nu) : R^{(L)} \in \text{TREE}^\lambda\{y'(\phi, \nu)\}\right\}$, it remains to characterize the set of $\phi \in S_{grow}(\mathcal{B}, \nu)$ such that $R^{(L)}$ is not removed during pruning. Recall that $g(\cdot)$ was defined in (2.3).

Proposition 2.4.3 *There exists a tree $\text{TREE}(\mathcal{B}, \nu, \lambda)$ such that*

$$S^\lambda(\mathcal{B}, \nu) = S_{grow}(\mathcal{B}, \nu) \cap \left(\bigcap_{l=0}^{L-1} \left\{\phi : g\left\{R^{(l)}, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu)\right\} \geq \lambda\right\} \right). \tag{2.21}$$

If $\mathcal{R}(\mathcal{B}) \in \text{TREE}^\lambda(y)$, then $\text{TREE}(\mathcal{B}, \nu, \lambda) = \text{TREE}^\lambda(y)$ satisfies (2.21). *Otherwise, given the set $S_{grow}(\mathcal{B}, \nu)$, computing a $\text{TREE}(\mathcal{B}, \nu, \lambda)$ that satisfies* (2.21) *has a worst-case computational cost of $O(n^2 p)$.*

We explain how to compute a $\text{TREE}(\mathcal{B}, \nu, \lambda)$ satisfying (2.21) when $\mathcal{R}(\mathcal{B}) \notin \text{TREE}^\lambda(y)$ in the supplementary materials.

Proposition 2.4.4 *The set $\bigcap_{l=0}^{L-1} \left\{\phi : g\left\{R^{(l)}, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu)\right\} \geq \lambda\right\}$ in* (2.21) *is the intersection of the solution sets of L quadratic inequalities in ϕ. Given $\text{TREE}(\mathcal{B}, \nu, \lambda)$, the coefficients of these quadratics can be obtained in $O(nL)$ operations. After $S_{grow}(\mathcal{B}, \nu)$ has been computed, intersecting it with these quadratic sets to obtain $S^\lambda(\mathcal{B}, \nu)$ from* (2.21) *requires $O\{npL \times \log(npL)\}$ operations in general, and only $O(L)$ operations if $\mathcal{B} = \text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}$ and $\nu = \nu_{sib}$ from* (2.6).

The results in this section have relied upon Condition 2.1. Indeed, this condition holds for branches $\mathcal{B}$ and vectors ν that arise in characterizing the sets $S^\lambda_{sib}(\nu_{sib})$ and $S^\lambda_{reg}(\nu_{reg})$.

Proposition 2.4.5 *If either (i)* $\mathcal{B} = \textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}$ *and* $\nu = \nu_{sib}$ (2.6)*, where* R_A *and* R_B *are siblings in* $\textsc{tree}^\lambda(y)$*, or (ii)* $\mathcal{B}$ *is a permutation of* $\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}$ *and* $\nu = \nu_{reg}$ (2.14)*, where* $R_A \in \textsc{tree}^\lambda(y)$*, then Condition 2.1 holds.*

Combining Lemma 2.1 with Propositions 2.4.1–2.4.5, we see that $S^\lambda_{sib}(\nu_{sib})$ can be computed in $O\{npL + np\log(n)\}$ operations. However, computing $S^\lambda_{reg}(\nu_{reg})$ is much more computationally intensive: by Lemma 2.2 and Propositions 2.4.1–2.4.5, it requires computing $S^\lambda(\pi\left[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}\right], \nu_{reg})$ for all $L!$ permutations $\pi \in \Pi$, for a total of $O\left[L!\left\{n^2 pLlog(pL)\right\}\right]$ operations. In Section 2.4.3, we discuss ways to avoid these calculations.

2.4.3 A computationally-efficient alternative to $S^\lambda_{reg}(\nu_{reg})$

Lemma 2.2 suggests that carrying out inference on a single region requires computing $S^\lambda(\pi\left[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}\right], \nu_{reg})$ for every $\pi \in \Pi$. We now present a less computationally demanding alternative.

Proposition 2.4.6 *Let* Q *be a subset of the* $L!$ *permutations in* Π*, i.e.* $Q \subseteq \Pi$*. Define*

$$p^Q_{reg}(y) = pr_{H_0}\left\{|\nu^{\mathrm{T}}_{reg}Y - c| \geq |\nu^{\mathrm{T}}_{reg}y - c| \mid \bigcup_{\pi\in Q}\left(\mathcal{R}(\pi[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}]) \subseteq \textsc{tree}^\lambda(Y)\right), \mathcal{P}^\perp_{\nu_{reg}}Y = \mathcal{P}^\perp_{\nu_{reg}}y\right\}.$$

The test based on $p^Q_{reg}(y)$ *controls the selective Type 1 error rate* (2.4) *for* $H_0 : \nu^{\mathrm{T}}_{reg}\mu = c$*. Furthermore,* $p^Q_{reg}(y) = pr\left\{|\phi - c| \geq |\nu^{\mathrm{T}}_{reg}y - c| \mid \phi \in \bigcup_{\pi\in Q} S^\lambda\left(\pi[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}], \nu_{reg}\right)\right\}$*, where* $\phi \sim N(c, \|\nu_{reg}\|_2^2\sigma^2)$*.*

Using the notation in Proposition 2.4.6, $p_{reg}(y)$ introduced in (2.15) equals $p^\Pi_{reg}(y)$. If we take $Q = \{\mathcal{I}\}$, where $\mathcal{I}$ is the identity permutation, then we arrive at

$$p^{\mathcal{I}}_{reg}(y) = P\left(|\phi - c| \geq |\nu^{\mathrm{T}}_{reg}y - c| \mid \phi \in S^\lambda[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}, \nu_{reg}]\right), \tag{2.22}$$

where $\phi \sim N(c, \|\nu_{reg}\|_2^2\sigma^2)$. The set $S^\lambda[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}, \nu_{reg}]$ can be easily computed by Proposition 2.4.3.

Compared to (2.15), (2.22) conditions on an extra piece of information: the ancestors of R_A. Thus, while (2.22) controls the selective Type 1 error rate, it may have lower power than (2.15) [Fithian et al., 2014]. Similarly, inverting (2.22) to form a confidence interval provides correct selective coverage, but may yield intervals that are wider than those in Proposition 2.3.2. Proposition 2.4.6 is motivated by a proposal by Lee et al. [2016] to condition on both the selected model (necessary information) and the signs of the selected variables (extra information) in the lasso setting, to gain computational efficiency at the possible expense of precision and power.

In Appendix A.6, we show through simulation that the loss in power associated with using (2.22) rather than (2.15) is negligible. Thus, in practice, we suggest using (2.22) for its computational efficiency. We use (2.22) for the remainder of this chapter.

Furthermore, we can consider computing confidence intervals of the form $[L_{S^{\mathcal{I}}_{reg}}(y), U_{S^{\mathcal{I}}_{reg}}(y)]$ rather than (2.17), where $L_{S^{\mathcal{I}}_{reg}}(y)$ and $U_{S^{\mathcal{I}}_{reg}}(y)$ satisfy

$$\begin{aligned} F\left(\nu_{reg}^{\mathrm{T}} y; L_{S^{\mathcal{I}}_{reg}}(y), \sigma^2\|\nu_{reg}\|_2^2, S^\lambda\left[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}, \nu_{reg}\right]\right) &= 1 - \frac{\alpha}{2}, \\ F\left(\nu_{reg}^{\mathrm{T}} y; U_{S^{\mathcal{I}}_{reg}}(y), \sigma^2\|\nu_{reg}\|_2^2, S^\lambda\left[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}, \nu_{reg}\right]\right) &= \frac{\alpha}{2}. \end{aligned} \tag{2.23}$$

In Appendix A.6, we show that the confidence intervals resulting from (2.23) are not much wider than those resulting from (2.17). We therefore make use of confidence intervals of the form (2.23) in the remainder of this chapter.

2.5 Simulation study

2.5.1 Data generating mechanism

We simulate $X \in \mathbb{R}^{n\times p}$ with $n = 200, p = 10$, $X_{ij} \overset{i.i.d.}{\sim} N(0,1)$, and $y \sim N_n(\mu, \sigma^2 I_n)$ with $\sigma = 5$ and $\mu_i = b \times \left[1_{(x_{i,1}\leq 0)} \times \{1 + a1_{(x_{i,2}>0)} + 1_{(x_{i,3}\times x_{i,2}>0)}\}\right]$. This μ vector defines a three-level tree, shown in Figure 2.3 for three values of $a \in \mathbb{R}$.

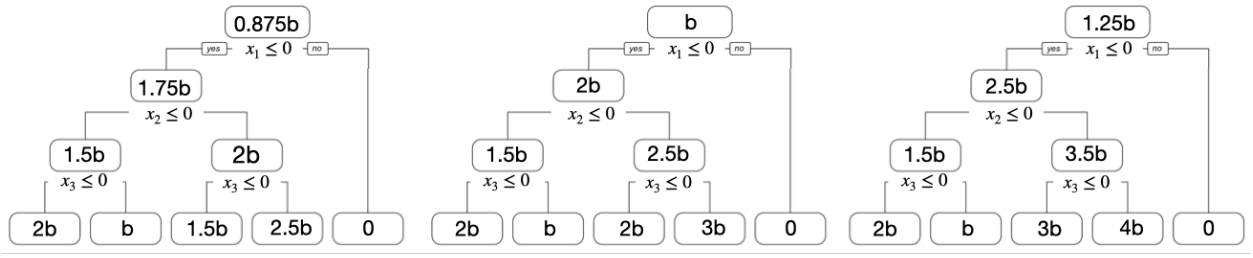


Figure 2.3: The true mean model in Section 2.5, for $a = 0.5$ (left), $a = 1$ (center), and $a = 2$ (right). The difference in means between the sibling nodes at level two in the tree is ab, while the difference in means between the sibling nodes at level three is b.

2.5.2 Methods for comparison

All CART trees are fit using the `R` package `rpart` [Therneau and Atkinson, 2019] with $\lambda = 200$, a maximum level of three, and a minimum node size of one. We compare three approaches for conducting inference. (i) *Selective Z-methods:* Fit a CART tree to the data. For each split, test for a difference in means between the two sibling regions using (2.8), and compute the corresponding confidence interval in (2.13). Compute the confidence interval for the mean of each region using (2.23). (ii) *Naive Z-methods:* Fit a CART tree to the data. For each split, conduct a naive Z-test for the difference in means between the two sibling regions, and compute the corresponding naive Z-interval. Compute a naive Z-interval for each region's mean. (iii) *Sample splitting:* Split the data into equally-sized training and test sets. Fit a CART tree to the training set. On the test set, conduct a naive Z-test for each split and compute a naive Z-interval for each split and each region. If a region has no test set observations, then we fail to reject the null hypothesis and fail to cover the parameter.

The conditional inference tree (CTree) framework of Hothorn et al. [2006] uses a different criterion than CART to perform binary splits. Within a region, it tests for linear association between each covariate and the response. The covariate with the smallest p-value for this linear association is selected as the split variable, and a Bonferroni corrected p-value that accounts for the number of covariates is reported in the final tree. Then, the split point

is selected. If, after accounting for multiple testing, no variable has a p-value below a pre-specified significance level α, then the recursion stops. While CTree's p-values assess linear association and thus are not directly comparable to the p-values in (i)–(iii) above, it is the most popular framework currently available for determining if a regression tree split is statistically significant. Thus, we also evaluate the performance of (iv) *CTree:* Fit a CTree to all of the data using the `R` package `partykit` [Hothorn and Zeileis, 2015] with $\alpha = 0.05$. For each split, record the p-value reported by `partykit`.

In Sections 2.5.3–2.5.6, we assume that σ is known. We consider the case of unknown σ in Section 2.5.7.

2.5.3 Uniform p-values under a Global Null

We generate $5,000$ datasets with $a = b = 0$, so that $H_0 : \nu_{sib}^{\mathrm{T}}\mu = 0$ holds for all splits in all trees. Figure 2.4 displays the distributions of p-values across all splits in all fitted trees for the naive Z-test, sample splitting, and the selective Z-test. The selective Z-test and sample splitting achieve uniform p-values under the null, while the naive Z-test (which does not account for the fact that ν_{sib} was obtained by applying CART to the same data used for testing) does not. CTree is omitted from the comparison: it creates a split only if the p-value is less than $\alpha = 0.05$, and thus its p-values over the splits do not follow a Uniform(0,1) distribution.

2.5.4 Power

We generate 500 datasets for each $(a, b) \in \{0.5, 1, 2\} \times \{1, \ldots, 10\}$, and evaluate the power of selective Z-tests, sample splitting, and CTree to reject the null hypothesis $H_0 : \nu_{sib}^{\mathrm{T}}\mu = 0$. As naive Z-tests do not control the Type 1 error rate (Figure 2.4), we do not evaluate their power. We consider two aspects of power: the probability that we *detect* a true split, and the probability that we *reject* the null hypothesis corresponding to a true split.

Given a true split in Figure 2.3 and an estimated split, we construct the 3×3 contingency table in Table 2.1, which indicates whether an observation is on the left-hand side, right-

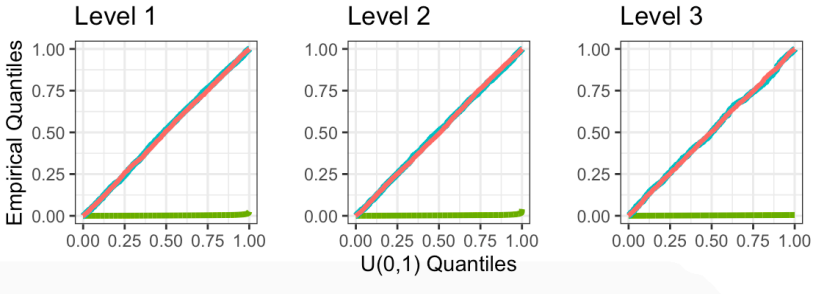


Figure 2.4: Quantile-quantile plots of the p-values for testing $H_0 : \nu_{sib}^{\mathrm{T}} \mu = 0$, as described in Section 2.5.3. A naive Z-test (green), sample splitting (blue), and selective Z-test (pink) were performed; see Section 2.5.2. The p-values are stratified by the level of the regions in the fitted tree.

Table 2.1: A 3×3 contingency table indicating an observation's involvement in a given true split and estimated split. The adjusted Rand index is computed using only the shaded cells

		Estimated Split		
		In left region	In right region	In neither
True Split	In left region	t_1	t_2	t_3
	In right region	u_1	u_2	u_3
	In neither	v_1	v_2	v_3

hand side, or not involved in the true split (rows) and the estimated split (columns). To quantify the agreement between the true and estimated splits, we compute the adjusted Rand index [Hubert and Arabie, 1985] associated with the 2×3 contingency table corresponding to the shaded region in Table 2.1. For each true split, we identify the estimated split for which the adjusted Rand index is largest; if this index exceeds 0.75 then this true split is "detected". Given that a true split is detected, the associated null hypothesis is rejected if the corresponding p-value is below 0.05. Figure 2.5 displays the proportion of true splits that are detected and rejected by each method.

As sample splitting fits a tree using only half of the data, it detects fewer true splits, and thus rejects the null hypothesis for fewer true splits, than the selective Z-test.

When a is small, the difference in means between sibling regions at level two is small. Because CTree makes a split only if there is strong evidence of association at that level, it tends to build one-level trees, and thus fails to detect many true splits; by contrast, the selective Z-test (based on CART) successfully builds more three-level trees. Thus, when a is small, the selective Z-test detects (and rejects) more true differences than CTree between regions at levels two and three.

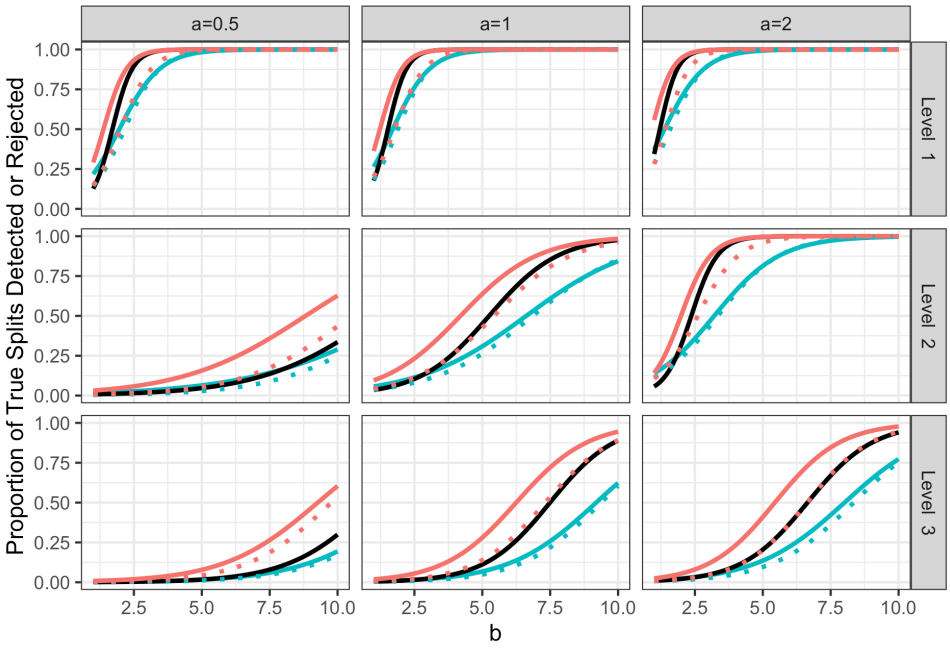


Figure 2.5: Proportion of true splits detected (solid lines) and rejected (dotted lines) for CART with selective Z-tests (pink), CTree (black), and CART with sample splitting (blue) across different settings of the data generating mechanism, stratified by level in tree. As CTree only makes a split if the p-value is less than 0.05, the proportion of detections equals the proportion of rejections.

2.5.5 Coverage of confidence intervals for $\nu_{sib}^{\mathrm{T}}\mu$ and $\nu_{reg}^{\mathrm{T}}\mu$

We generate 500 datasets for each $(a, b) \in \{0.5, 1, 2\} \times \{0, \ldots, 10\}$ to evaluate the coverage of 95% confidence intervals constructed using naive Z-methods, selective Z-methods, and

sample splitting. CTree is omitted from these comparisons because it does not provide confidence intervals. We say that the interval covers the truth if it contains $\nu^{\mathrm{T}}\mu$, where ν is defined as in (2.6) (for a particular split) or (2.14) (for a particular region). Table 2.2 shows the proportion of each type of interval that covers the truth, aggregated across values of a and b. The selective Z-intervals attain correct coverage of 95%, while the naive Z-intervals do not.

It may come as a surprise that sample splitting does not attain correct coverage. Recall that ν from (2.6) or (2.14) is an n-vector that contains entries for all observations in both the training set and the test set. Thus, $\nu^{\mathrm{T}}\mu$ involves the true mean among both training and test set observations in a given region or pair of regions. By contrast, sample splitting attains correct coverage for a different parameter involving the true means of only the test observations that fall within a given region or pair of regions.

2.5.6 Width of confidence intervals

Figure 2.6(a) illustrates that our selective Z-intervals for $\nu_{reg}^{\mathrm{T}}\mu$ can be extremely wide when b is small, particularly for regions located at deeper levels in the tree. For each tree that we build and for levels 1, 2, and 3, we compute the adjusted Rand Index [Hubert and Arabie, 1985] between the true tree (truncated at the appropriate level) and the estimated tree (truncated at the same level). Figure 2.6(b) shows that our selective confidence intervals can be extremely wide when this adjusted Rand Index is small, particularly at deeper levels of the tree.

When b is small and the adjusted Rand Index is small, the trees built by CART tend to be unstable, in the sense that small perturbations to the data affect the fitted tree. In this setting, the sample statistics $\nu_{reg}^{\mathrm{T}}y$ fall very close to the boundary of the truncation set. See Kivaranovic and Leeb [2021] for a discussion of why wide confidence intervals can arise in these settings. The great width of our confidence intervals reflects the uncertainty about the mean response within each region due to the instability of the tree-fitting procedure.

Table 2.2: Proportion of 95% confidence intervals containing the true parameter, aggregated over all trees fit to the 5,500 datasets generated with $(a, b) \in \{0.5, 1, 2\} \times \{1, \ldots, 10\}$.

	Parameter $\nu_{reg}^{\mathrm{T}}\mu$			Parameter $\nu_{sib}^{\mathrm{T}}\mu$		
Level	Selective Z	Naive Z	Sample Splitting	Selective Z	Naive Z	Sample Splitting
1	0.951	0.889	0.918	0.948	0.834	0.915
2	0.950	0.645	0.921	0.951	0.410	0.917
3	0.951	0.711	0.921	0.950	0.550	0.921

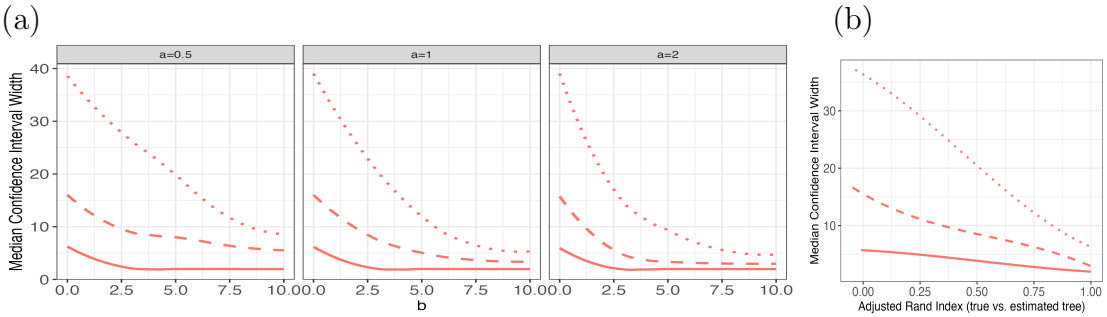


Figure 2.6: The median width of the selective Z-intervals for parameter $\nu_{reg}^{\mathrm{T}}\mu$ for regions at levels one (solid), two (dashed), and three (dotted) of the tree. Similar results hold for parameter $\nu_{sib}^{\mathrm{T}}\mu$. Panel (a) breaks results down by the parameters a and b, whereas panel (b) aggregates results across values of parameters a and b, and displays them as a function of the adjusted Rand Index between the true and estimated trees.

2.5.7 Results with unknown σ

Thus far, we have assumed that σ is known. In this section, we compare the following three versions of the selective Z-methods that plug different values of σ into the truncated normal CDF when computing p-values and confidence intervals:

1. σ: We plug in the true value of σ, as in Sections 2.5.3–2.5.6.

2. $\hat{\sigma}_{\mathrm{cons}}$: We plug in $\hat{\sigma}_{\mathrm{cons}} = \sqrt{(n-1)^{-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}$, where $\bar{y} = n^{-1}\sum_{i=1}^{n} y_i$.

3. $\hat{\sigma}_{\mathrm{SSE}}$: Let $\mathcal{T} = \left|\textsc{term}\left(\mathbb{R}^p, \textsc{tree}^{\lambda}(y)\right)\right|$ be the number of terminal regions in $\textsc{tree}^{\lambda}(y)$.

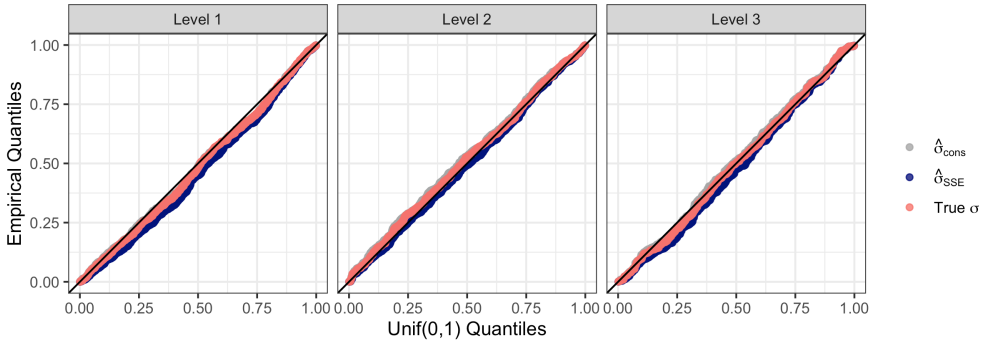


Figure 2.7: QQ plots of the p-values from testing $H_0 : \nu_{sib}^T \mu = 0$ when $\mu = 0_n$ using the selective Z-test with three different values plugged in to the truncated normal CDF for σ. The p-values are stratified by the level of the regions in the fitted tree.

We plug in $\hat{\sigma}_{\text{SSE}} = \sqrt{(n - \mathcal{T})^{-1} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$, where $\hat{y}_i$ is the predicted value for the ith observation given by $\textsc{Tree}^{\lambda}(y)$.

It is straightforward to show that $E[\hat{\sigma}^2_{\text{cons}}] \geq \sigma^2$, for any value of $E[y] = \mu$. Thus, we expect this estimate to lead to conservative inference. On the other hand, $\hat{\sigma}^2_{\text{SSE}}$ can be made arbitrarily small by making the fitted tree arbitrarily deep, and so we expect inference based on this estimate to be anti-conservative if the fitted CART tree is large.

Figure 2.7 shows the distribution of p-values from testing $H_0 : \nu_{sib}^T \mu = 0$ with the three versions of the selective Z-test under the data generating mechanism described in Section 5.1, with $a = b = 0$. In this setting, $\nu_{sib}^T \mu = 0$ holds for all splits in all trees. We see almost no difference between the three versions of the selective Z-test. In this global null setting, $E[\hat{\sigma}^2_{\text{cons}}] = \sigma^2$. Furthermore, the empirical bias of $\hat{\sigma}^2_{\text{SSE}}$ is small because the trees we grow are not particularly large; as in the rest of Section 2.5, we build trees to a maximum depth of 3 and prune with $\lambda = 200$.

Figure 2.8 displays the proportion of true splits detected and the proportion of true splits

detected and rejected, as defined in Section 2.5.4, for the three versions of the selective Z-test when data is generated as in Section 2.5.4. For simplicity, we only show the setting where $a = 1$. All three methods detect the same proportion of true splits, because they all perform inference on the same CART trees. The proportion of splits detected and rejected is very similar for σ and $\hat{\sigma}_{\text{SSE}}$ because $\hat{\sigma}_{\text{SSE}}$ is a very good estimator for σ in this setting. While $\hat{\sigma}_{\text{cons}}$ performs reasonably when b is small, it severely overestimates σ and thus has low power when b is large.

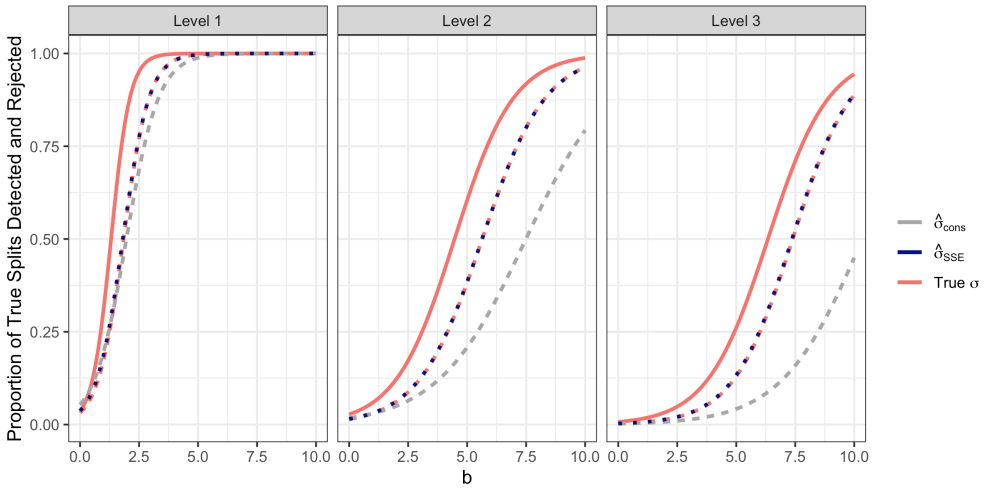


Figure 2.8: Proportion of true splits detected (solid lines) and rejected (dotted lines) for CART with the three versions of the selective Z-test. The results are stratified by level in tree.

Table 2.3 displays confidence intervals for $\nu_{sib}^{\text{T}}\mu$ and $\nu_{reg}^{\text{T}}\mu$ for the three versions of the selective Z-intervals, where data is generated as in Section 2.5.5. As expected, $\hat{\sigma}_{\text{cons}}$ leads to slight over-coverage and $\hat{\sigma}_{\text{SSE}}$ leads to slight under-coverage.

	Parameter $\nu_{reg}^{\mathrm{T}}\mu$			Parameter $\nu_{sib}^{\mathrm{T}}\mu$		
Level	σ	$\hat{\sigma}_{\mathrm{cons}}$	$\hat{\sigma}_{\mathrm{SSE}}$	σ	$\hat{\sigma}_{\mathrm{cons}}$	$\hat{\sigma}_{\mathrm{SSE}}$
1	0.95	0.98	0.95	0.95	0.98	0.94
2	0.95	0.97	0.94	0.95	0.97	0.94
3	0.95	0.96	0.94	0.95	0.96	0.94

Table 2.3: Proportion of 95% confidence intervals containing the true parameter, aggregated over all trees fit to the 5,500 datasets generated with $(a, b) \in \{0.5, 1, 2\} \times \{1, \ldots, 10\}$

In this section, we have seen that when trees are not grown overly large, plugging in $\hat{\sigma}_{\mathrm{SSE}}$ leads to approximate selective Type 1 error control, approximately correct selective coverage, and good power. Unfortunately, providing theoretical guarantees for our procedures when using $\hat{\sigma}_{\mathrm{SSE}}$ would be quite difficult, as the estimator is anti-conservative and depends on the output of CART. Providing theoretical guarantees for our procedures under $\hat{\sigma}_{\mathrm{cons}}$ is more straightforward, using ideas from Gao et al. [2022], Chen and Witten [2023], and Tibshirani et al. [2018]. However, as shown in Figure 2.8, selective Z-tests based on $\hat{\sigma}_{\mathrm{cons}}$ can have very low power. One promising avenue of future work involves providing theoretical guarantees in the regression tree setting for estimators that are less conservative than $\hat{\sigma}_{\mathrm{cons}}$.

2.6 An application to the Box Lunch Study

Venkatasubramaniam et al. [2017] compare CART and CTree [Hothorn et al., 2006] within the context of epidemiological studies. They conclude that CTree is preferable to CART because it provides p-values for each split, even though CART has higher predictive accuracy. Since our framework provides p-values for each split in a CART tree, we revisit their analysis of the Box Lunch Study, a clinical trial studying the impact of portion control interventions on 24-hour caloric intake. We consider identifying subgroups of study participants with baseline differences in 24-hour caloric intake on the basis of scores from an assessment that quantifies constructs such as hunger, liking, the relative reinforcement of food (`rrvfood`), and restraint (`resteating`).

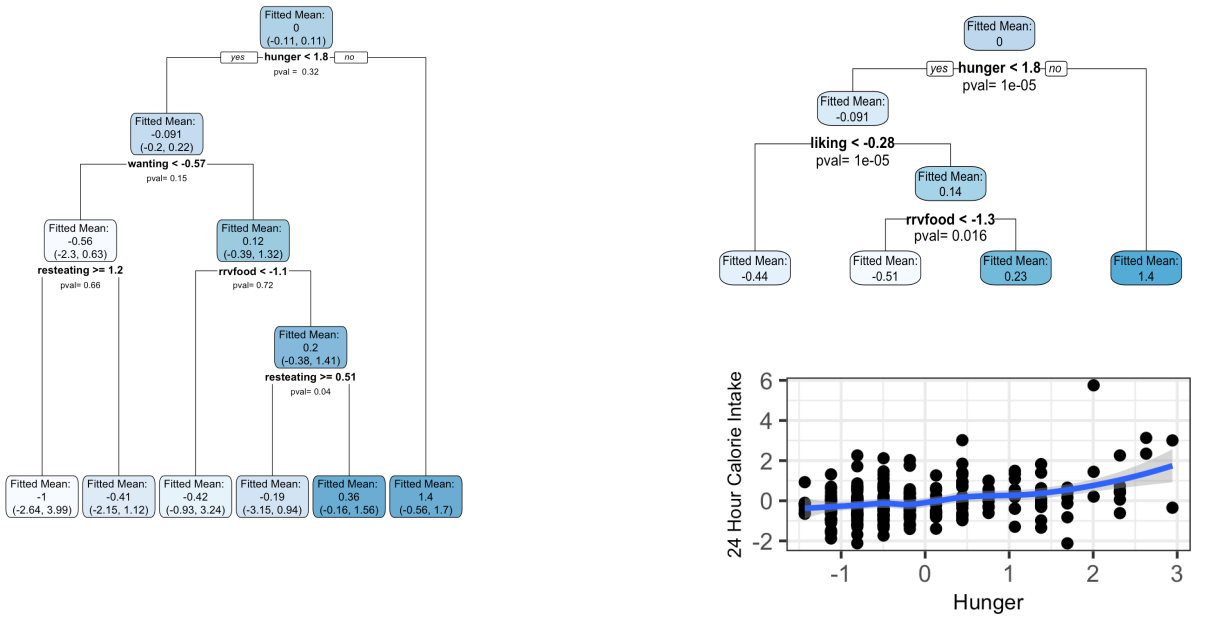


Figure 2.9: *Left:* A CART tree fit to the Box Lunch Study data. Each split has been labeled with a p-value (2.8), and each region has been labeled with a confidence interval (2.23). The shading of the nodes indicates the average response values (white indicates a very small value and dark blue a very large value). *Top right:* A CTree fit to the Box Lunch Study data. *Bottom right:* A scatterplot showing the relationship between the covariate hunger and the response.

We exactly reproduce the trees presented in Figures 1 and 2 of Venkatasubramaniam et al. [2017] by building a CTree using `partykit` and a CART tree using `rpart` on the Box Lunch Study data provided in the `R` package `visTree` [Venkatasubramaniam and Wolfson, 2018]. We apply our selective inference framework to compute p-values (2.8) for each split in CART, and confidence intervals (2.23) for each region. In this section, we use $\hat{\sigma}_{\text{SSE}}$, defined in Section 2.5.7, to estimate the error variance. The results are shown in Figure 2.9.

Both CART and CTree choose `hunger`$<$`1.8` as the first split. For this split, our selective Z-test reports a large p-value of 0.44, while CTree reports a p-value less than 0.001. The conflicting p-values are explained by the difference in null hypotheses. CTree finds strong

evidence against the null of no linear association between `hunger` and caloric intake. By contrast, our selective framework for CART does not find strong evidence for a difference between mean caloric intake of participants with `hunger`$<$`1.8` and those with `hunger`$\geq$`1.8`. We see from the bottom right of Figure 2.9 that while there is evidence of a linear relationship between `hunger` and caloric intake, there is less evidence of a difference in means across the particular split `hunger`=`1.8`. Given that the goal of Venkatasubramaniam et al. [2017] is to "identify population subgroups that are relatively homogeneous with respect to an outcome", the p-value resulting from our selective framework is more natural than the p-value output by CTree, since the former relates directly to the subgroups formed by the split, whereas the latter does not take into account the location of the split point. In general, the left-hand panel of Figure 2.9 shows that the subgroups of patients identified by CART are not significantly different from one another. This is an important finding that would be missed without our selective inference framework. Furthermore, unlike CTree, our framework provides confidence intervals for the mean response in each subgroup.

An alternative analysis using $\hat{\sigma}_{\text{cons}}$, defined in Section 2.5.7, is provided in Appendix A.8, and leads to similar findings.

2.7 Discussion

Our framework relies on the assumption that $Y \sim N_n(\mu, \sigma^2 I)$, with σ^2 known. In Section 2.5.7, we showed strong empirical performance when the variance is unknown and σ^2 is estimated. In this section, we briefly comment on the assumptions of spherical variance and normally distributed data.

It natural to wonder whether the assumption that $Y \sim N_n(\mu, \sigma^2 I)$ can be relaxed to the assumption that $Y \sim N_n(\mu, \Sigma)$, with Σ known. Following the work of Lee et al. [2016], the results in Section 2.3 extend to the setting where $Y \sim N_n(\mu, \Sigma)$ if we:

1. Modify (2.8) and (2.15) to condition on the event $\left\{\left(I_n - \frac{\Sigma\nu\nu^T}{\nu^T\Sigma\nu}\right) Y = \left(I_n - \frac{\Sigma\nu\nu^T}{\nu^T\Sigma\nu}\right) y\right\}$ rather than the event $\{\mathcal{P}_\nu^\perp Y = \mathcal{P}_\nu^\perp y\}$, where $\nu = \nu_{sib}$ in the case of (8) and $\nu = \nu_{reg}$ in

the case of (15).

2. Replace all instances of the perturbation $y'(\phi, \nu)$, defined in Theorem 2.1, with the perturbation $y''(\phi, \nu) = \left(I_n - \frac{\Sigma\nu\nu^T}{\nu^T\Sigma\nu}\right) y + \frac{\Sigma\nu}{\nu^T\Sigma\nu}\phi$.

Unfortunately, the modified perturbation $y''(\phi, \nu)$ does not satisfy Condition 2.1 in Section 2.4.2 when $\Sigma \neq \sigma^2 I_n$, and so many of the results of Section 2.4 do not extend to this non-spherical setting. Future work could explore how to efficiently compute the conditioning set in this non-spherical setting.

Furthermore, our framework assumes a normally-distributed response variable. CART is commonly used for classification, survival [Segal, 1988], and treatment effect estimation in causal inference [Athey and Imbens, 2016]. While the idea of conditioning on a selection event to control the selective Type 1 error rate applies regardless of the distribution of the response, our Theorem 2.1 and Theorem 2.2, and the resulting computational results, relied on normality of Y. In the absence of this assumption, exactly characterizing the conditioning set and the distribution of the test statistic requires further investigation.

We show in Appendix A.7 that our selective Z-tests approximately control the selective Type 1 error when the normality assumption is violated. Tian and Taylor [2017] and Tibshirani et al. [2018] establish conditions under which selective p-values for linear regression (derived under the assumption of normality) will be asymptotically uniformly distributed under non-normality. Thus suggests the possibility of developing asymptotic theory for our proposed selective Z-tests under violations of normality.

A reviewer pointed out similarities between the problem of testing significance of the first split in the tree and significance testing for a single changepoint, as in Bhattacharya [1994]. Building on this connection may provide an avenue for future work.

A software implementation of the methods in this chapter is available in the `R` package `treevalues`, at `https://github.com/anna-neufeld/treevalues`.

Chapter 3

INFERENCE AFTER LATENT VARIABLE ESTIMATION FOR SINGLE-CELL RNA SEQUENCING DATA

The contents of this chapter are published in *Biostatistics* [Neufeld et al., 2022a].

3.1 Introduction

Techniques for single-cell RNA sequencing (scRNA-seq) allow scientists to measure gene expression of huge numbers of individual cells in parallel. Researchers can then investigate how gene expression varies between cells of different states. In particular, we highlight two common questions that arise in the context of scRNA-seq data:

> **Question 1.** *Which genes are differentially expressed along a continuous cellular trajectory?* This trajectory might represent development, activity level of an important pathway, or *pseudotime*, a quantitative measure of biological progression through a process such as cell differentiation [Trapnell et al., 2014].
>
> **Question 2.** *Which genes are differentially expressed between discrete cell types?*

These two questions are hard to answer because typically the cellular trajectory or the cell types are not directly observed, and must be estimated from the data. We can unify these two questions, and many others that arise in the analysis of scRNA-seq data, under a latent variable framework.

Suppose that we have mapped the scRNA-seq reads for n cells to p genes (or other functional units of interest). Then, the data matrix X has dimension $n \times p$, and X_{ij} is the number of reads from the ith cell that map to the jth gene. We assume that X is a realization of a random variable $\mathbf{X}$, and that the biological variation in $\mathrm{E}[\mathbf{X}]$ is explained by

a set of latent variables $L \in \mathbb{R}^{n \times k}$. We wish to know which columns $\mathbf{X}_j$ of $\mathbf{X}$ are associated with L. As L is unobserved, the following two-step procedure seems natural:

Step 1: Latent variable estimation. Use X to compute $\widehat{L}(X)$, an estimate of L.

Step 2: Differential expression analysis. For $j = 1, \ldots, p$, test for association between $\mathbf{X}_j$ and the columns of $\widehat{L}(X)$.

In this chapter, we refer to the practice of using the same data X to first construct $\hat{L}(X)$ and second to test for association with $\hat{L}(X)$ as "double dipping".

Why is double dipping a problem? As we will see throughout this chapter, if we use standard statistical tests in Step 2, then we will fail to control the Type 1 error rate. We now provide intuition for this using a very simple motivating example. We generate $X \in \mathbb{Z}_{\geq 0}^{500 \times 5}$, where $\mathbf{X}_{ij} \sim \text{Poisson}(5)$ for $i = 1, \ldots, 500$ and $j = 1, \ldots, 5$. The distribution of counts for each of the five genes is shown in the left column of Figure 3.1. The right panel of Figure 3.1 shows the result of (1) clustering the data by applying k-means with $k = 2$, and then (2) testing for association between each gene and the estimated clusters (using a Wald p-value from a Poisson generalized linear model). Even though all cells are drawn from the same distribution, and thus all tested null hypotheses hold, all five p-values are small. The Type 1 error rate is not controlled, because the Wald test does not account for the fact that the clustering algorithm is designed to maximize the difference between the clusters (see the right panel of Figure 3.1).

Despite this issue with double dipping, it is common practice. `Monocle3` and `Seurat` are popular `R` packages that each contain functions for (1) estimating latent variables such as clusters or pseudotime, and (2) identifying genes that are differentially expressed across these latent variables. The vignettes for these `R` packages perform both steps on the same data [Pliner et al., 2022, Hoffman et al., 2022]. Consequently, the computational pipelines suggested by the package vignettes fail to control the Type 1 error rate. We demonstrate this empirically in Appendix A of the supplementary materials.

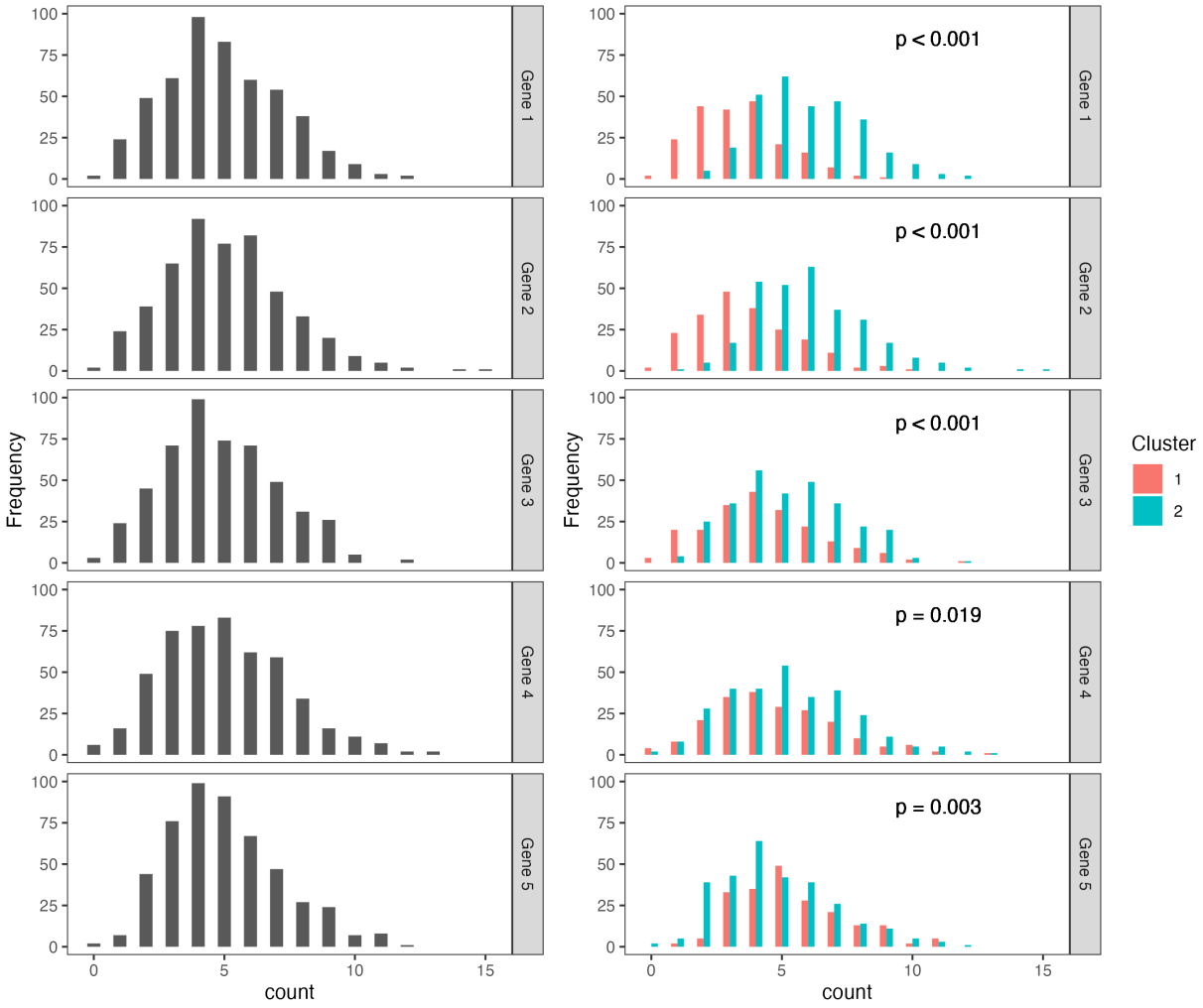


Figure 3.1: *Left:* Distributions of counts for five genes under a model where $\mathbf{X}_{ij} \sim$ Poisson(5) for all genes and all cells. *Right:* Distributions of the same counts, colored by estimated cluster, labeled with the Wald p-values from a Poisson GLM. All p-values are small, despite the fact that all null hypotheses hold.

Though no suitable solution has yet been proposed, the issues associated with this two-step process procedure are well-documented: Lähnemann et al. [2020] cite the "double use of data" in differential expression analysis after clustering as one of the "grand challenges" in single cell RNA sequencing (Question 1), and Deconinck et al. [2021] note that the "circularity" involved in the two-step process of trajectory analysis leads to "artificially low p-values" for differential expression (Question 2).

In this chapter, we propose *count splitting*, a simple fix for the double dipping problem. This allows us to carry out latent variable estimation and differential expression analysis without double dipping, so as to obtain p-values that control the Type 1 error for the null hypothesis that a given gene is not associated with an estimated latent variable.

In Section 3.2, we introduce the notation and models that will be used in this chapter. In Section 3.3, we carefully examine existing methods for latent variable inference, and explain why they are not adequate in this setting. In Section 3.4 we introduce our proposed method, and in Sections 3.5 and 3.6 we demonstrate its merits on simulated and real data.

3.2 Models for scRNA-seq data

Recall that X_{ij} is the number of reads mapping to the jth gene in the ith cell, and that $X \in \mathbb{Z}_{\geq 0}^{n \times p}$ is a realization from $\mathbf{X}$. We assume that the entries in $\mathbf{X}$ are independent, with

$$\mathrm{E}\left[\mathbf{X}_{ij}\right] = \gamma_i \Lambda_{ij}, \quad \log\left(\Lambda_{ij}\right) = \beta_{0j} + \beta_{1j}^T L_i, \quad i = 1, \ldots, n, \quad j = 1, \ldots, p, \tag{3.1}$$

where $\gamma_1, \ldots, \gamma_n$ are cell-specific size factors that reflect technical variation in capture efficiency of the mRNA molecules between cells, the $n \times p$ matrix Λ represents biological variation, and the matrix $L \in \mathbb{R}^{n \times k}$ contains the unobserved latent variables. In (3.1), $\beta_{1j} \in \mathbb{R}^k$ and L_i is the ith row of L. Throughout this chapter, we treat L, and thus Λ, as fixed. Estimating $\gamma_1, \ldots, \gamma_n$ can be challenging. As size factor estimation is not the focus of this chapter, we assume throughout that the γ_i are either known or have been accurately estimated.

Based on (3.1), in this chapter we carry out the differential expression analysis step of the two-step process introduced in Section 3.1 by fitting a generalized linear model (GLM) with a log link function to predict X_j using $\widehat{L}(X)$, with the γ_i included as offsets. We note that while the notation introduced in the remainder of this section is specific to GLMs, the ideas in this chapter can be extended to accommodate alternate methods for differential expression analysis, such as generalized additive models (as in Van den Berge et al. 2020 and Trapnell et al. 2014).

Under the model in (3.1), the columns of the matrix $\log(\Lambda)$ are linear combinations of the unobserved columns of L. However, they may not be linear combinations of the estimated latent variables, i.e. the columns of $\widehat{L}(X)$. Thus, we need additional notation to describe the population parameters that we target when we fit a GLM with $\widehat{L}(X)$, rather than L, as the predictor.

Let $p_\theta(\cdot)$ denote the density function of the distribution belonging to the exponential family specified by the GLM with mean θ. For any $Z \in \mathbb{R}^{n\times k}$ and random variable $\mathbf{X} \in \mathbb{R}^{n\times p}$, we define the population parameters targeted by fitting a GLM with a log link function to predict X_j (drawn from $\mathbf{X}_j$) using Z, with the γ_i included as offsets, to be:

$$\left(\beta_0(Z,\mathbf{X}_j),\beta_1(Z,\mathbf{X}_j)\right) = \underset{\alpha_0\in\mathbb{R},\alpha_1\in\mathbb{R}^k}{\arg\max}\left(\mathrm{E}_{\mathbf{X}_{1j},\ldots,\mathbf{X}_{nj}}\left[\sum_{i=1}^{n}\log\left(p_{\gamma_i\exp\left(\alpha_0+\alpha_1^T Z_i\right)}(\mathbf{X}_{ij})\right)\right]\right), \tag{3.2}$$

where the expectation in (3.2) is taken over the true joint distribution of $\mathbf{X}_{1j},\ldots,\mathbf{X}_{nj}$. We say that the jth gene is differentially expressed across a variable Z if $\beta_1(Z,\mathbf{X}_j) \neq 0$. If the mean model in (3.1) holds and the distribution of $\mathbf{X}_{ij}$ belongs to the family specified by the GLM for $i = 1,\ldots,n$, then $\beta_1(L,\mathbf{X}_j) = \beta_{1j}$ and $\beta_0(L,\mathbf{X}_j) = \beta_{0j}$ from (3.1). More generally, $\beta_0(Z,\mathbf{X}_j)$ and $\beta_1(Z,\mathbf{X}_j)$ are the parameters that make $\prod_{i=1}^{n} p_{\gamma_i\exp(\alpha_0+\alpha_1^T Z_i)}(X_{ij})$ closest in Kullback-Leibler (KL) divergence to the true joint distribution for $\mathbf{X}_{1j},\ldots,\mathbf{X}_{nj}$ (see Wakefield 2013, Section 2.4.3).

We denote the coefficient estimates that result from fitting a GLM with a log link function

to predict a realized X_j using Z, with the γ_i included as offsets, as

$$\left(\widehat{\beta}_0\left(Z, X_j\right), \widehat{\beta}_1\left(Z, X_j\right)\right) = \underset{\alpha_0 \in \mathbb{R}, \alpha_1 \in \mathbb{R}^k}{\arg\max} \sum_{i=1}^{n} \log\left(p_{\gamma_i \exp\left(\alpha_0 + \alpha_1^T Z_i\right)}\left(X_{ij}\right)\right). \tag{3.3}$$

For the majority of this chapter, as in Wang et al. [2018], we assume the model

$$\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} \text{Poisson}(\gamma_i \Lambda_{ij}), \tag{3.4}$$

and let $p_\theta(\cdot)$ be a Poisson density. In Section 3.4.2, we discuss the case where $\mathbf{X}_{ij}$ does not follow a Poisson distribution and/or $p_\theta(\cdot)$ is not a Poisson density.

3.3 Existing methods for latent variable inference

3.3.1 Motivating example

Throughout Section 3.3, we compare existing methods for differential expression analysis after latent variable estimation on a simple example. We generate $2,000$ realizations of $\mathbf{X} \in \mathbb{Z}_{\geq 0}^{200 \times 10}$, where $\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} \text{Poisson}(\Lambda_{ij})$. For $i = 1, \ldots, 200$, we let $\Lambda_{ij} = 1$ for $j = 1, \ldots, 5$ and $\Lambda_{ij} = 10$ for $j = 6, \ldots, 10$. (This is an example of the model defined in (4.1) and (3.4) with $\gamma_1 = \ldots = \gamma_n = 1$ and $\beta_{1j} = 0$ for $j = 1, \ldots, 10$.) Under this mechanism, each cell is drawn from the same distribution, and thus there is no true trajectory. Nevertheless, for each dataset X, we estimate a trajectory using the first principal component of the log-transformed data with a pseudocount of one. We then fit Poisson GLMs (with no size factors) to study differential expression. Since all columns of Λ are constant, $\beta_1(Z, \mathbf{X}_j) = 0$ for all j and for any $Z \in \mathbb{R}^n$ (see (3.2)). Therefore, any p-value quantifying the association between a gene and an estimated trajectory should follow a Unif$(0, 1)$ distribution. As we will see, most available approaches do not have this behavior.

3.3.2 The double dipping method

The *double dipping* method refers to using the same data for latent variable estimation and differential expression analysis, as in the two-step procedure in Section 3.1, without correcting for this double use. As mentioned in Section 3.1, variants of this method are used in the vignettes for the popular software packages `Seurat` [Hoffman et al., 2022] and `Monocle3` [Pliner et al., 2022]. In the context of the motivating example from Section 3.3.1, this method attempts to do inference on the parameter $\beta_1\left(\widehat{L}(X), \mathbf{X}_j\right)$, defined in (3.2), by regressing X_j on $\widehat{L}(X)$. Using notation from Section 3.2, the resulting Wald p-values have the form

$$\Pr_{H_0:\ \beta_1\left(\widehat{L}(X), \mathbf{X}_j\right)=0}\left(\left|\widehat{\beta}_1\left(\widehat{L}(X), \mathbf{X}_j\right)\right| \geq \left|\widehat{\beta}_1\left(\widehat{L}\left(X\right), X_j\right)\right|\right). \tag{3.5}$$

The right-hand side of the inequality in (3.5) uses the observed data to obtain both the predictor and the response, while the left-hand side does not. Thus, $\widehat{\beta}_1\left(\widehat{L}(X), X_j\right)$ is drawn from $\widehat{\beta}_1\left(\widehat{L}(\mathbf{X}), \mathbf{X}_j\right)$, *not* $\widehat{\beta}_1\left(\widehat{L}(X), \mathbf{X}_j\right)$. To state the issue in a different way, we used the data realization X to construct both the predictor $\hat{L}(X)$ and the response X_j in the GLM, and did not account for this double use in determining the distribution of the test statistic. Therefore, as shown in Figure 3.2(a), when we compute the p-value in (3.5) for many realizations of $\mathbf{X}$, the collection of p-values does not follow a Unif$(0, 1)$ distribution.

3.3.3 Cell splitting

In some settings, it is possible to overcome the issues associated with double dipping by splitting the observations into a training set and a test set, generating a hypothesis on the training set, and testing it on the test set [Cox, 1975]. However, in the setting of this chapter, splitting the cells in X does not allow us to bypass the issues of double dipping.

Why not? Suppose we estimate the latent variables using the cells in the training set, X^{train}. To test for differential expression using the cells in X^{test}, we need latent variable coordinates for the cells in X^{test}. However, it is not clear how to obtain latent variable coordinates for the cells in X^{test} that are only a function of the training set and not the

test set (i.e. can be written as $\widehat{L}(X^{\text{train}})$). In the simple example from Section 3.3.1, after computing the first principal axis of the log-transformed X^{train} matrix, we could consider projecting the log-transformed X^{test} onto this axis to obtain coordinates for the cells in X^{test}. Unfortunately, this projection step uses the data in X^{test}, and so the resulting estimated coordinates must be written as $\widehat{L}\left(X^{\text{train}}, X^{\text{test}}\right)$.

If we then fit a Poisson GLM to predict X_j^{test} using $\widehat{L}\left(X^{\text{train}}, X^{\text{test}}\right)$, the Wald p-values from this *cell splitting* procedure have the form

$$\Pr_{H_0:\ \beta_1\left(\widehat{L}(X^{\text{train}}, X^{\text{test}}), \mathbf{X}_j^{\text{test}}\right)=0}\left(\left|\widehat{\beta}_1\left(\widehat{L}(X^{\text{train}}, X^{\text{test}}), \mathbf{X}_j^{\text{test}}\right)\right| \geq \left|\widehat{\beta}_1\left(\widehat{L}\left(X^{\text{train}}, X^{\text{test}}\right), X_j^{\text{test}}\right)\right|\right). \tag{3.6}$$

Unfortunately, (3.6) suffers from the same issue as (3.5): the right-hand side of the inequality uses the same realization (X^{test}) to construct both the predictor and the response, whereas the left-hand side does not. Thus, the p-values from (3.6) do not follow a Unif$(0, 1)$ distribution, even when the columns of Λ are constants. Instead, they are anti-conservative, as shown in Figure 3.2(a).

3.3.4 Gene splitting

In the same spirit as cell splitting, we now consider splitting the genes (features), rather than the observations, to form X^{train} and X^{test}. In this setting, $\widehat{L}\left(X^{\text{train}}\right)$ provides coordinates for all cells in X, and we can obtain p-values for each gene X_j that is not in X^{train} by regressing X_j on $\widehat{L}(X^{\text{train}})$. The roles of X^{train} and X^{test} can be swapped to obtain p-values for the remaining genes. This *gene splitting* procedure yields Wald p-values of the form

$$\begin{aligned}
&\Pr_{H_0:\ \beta_1\left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)=0}\left(\left|\widehat{\beta}_1\left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)\right| \geq \left|\widehat{\beta}_1\left(\widehat{L}\left(X^{\text{train}}\right), X_j^{\text{test}}\right)\right|\right) \quad \text{if } X_j \in X^{\text{test}}, \\
&\Pr_{H_0:\ \beta_1\left(\widehat{L}(X^{\text{test}}), \mathbf{X}_j^{\text{train}}\right)=0}\left(\left|\widehat{\beta}_1\left(\widehat{L}(X^{\text{test}}), \mathbf{X}_j^{\text{train}}\right)\right| \geq \left|\widehat{\beta}_1\left(\widehat{L}\left(X^{\text{test}}\right), X_j^{\text{train}}\right)\right|\right) \quad \text{if } X_j \in X^{\text{train}}.
\end{aligned}$$

As the coefficients on the right-hand side of these inequalities never use the same data to construct the predictor and the response, these p-values will be uniformly distributed over repeated realizations of $\mathbf{X}$ when the columns of Λ are constants. However, they are

fundamentally unsatisfactory in scRNA-seq applications where we wish to obtain a p-value for every gene *with respect to the same estimated latent variable.* Thus, gene splitting is not included in Figure 3.2(a).

3.3.5 Selective inference through conditioning

We next consider taking a *selective inference* approach [Lee et al., 2016, Taylor and Tibshirani, 2015] to correct the p-values in (3.5). This involves fitting the same regression model as the method that double dips, but replacing (3.5) with the conditional probability

$$\Pr_{H_0:\ \beta_1\left(\widehat{L}(X),\mathbf{X}_j\right)=0}\left(\left|\widehat{\beta}_1\left(\widehat{L}(X),\mathbf{X}_j\right)\right| \geq \left|\widehat{\beta}_1\left(\widehat{L}\left(X\right),X_j\right)\right| \mid \widehat{L}\left(\mathbf{X}\right)=\widehat{L}(X)\right). \tag{3.7}$$

The inequality in (3.7) is identical to that in (3.5), but under the conditioning event, (3.7) can be rewritten as

$$\Pr_{H_0:\ \beta_1\left(\widehat{L}(X),\mathbf{X}_j\right)=0}\left(\left|\widehat{\beta}_1\left(\widehat{L}(\mathbf{X}),\mathbf{X}_j\right)\right| \geq \left|\widehat{\beta}_1\left(\widehat{L}\left(X\right),X_j\right)\right| \mid \widehat{L}\left(\mathbf{X}\right)=\widehat{L}(X)\right), \tag{3.8}$$

such that both the left-hand and right-hand sides of the inequality use the same data to construct the predictor and the response. Thus, over repeated realizations of $\mathbf{X}$ when the columns of Λ are constants, the p-values in (3.7) follow a Unif$(0,1)$ distribution.

The approach in (3.7) is not suitable for the setting of this chapter. First, (3.7) cannot be computed in practice. The selective inference literature typically modifies (3.7) by conditioning on extra information for computational tractability. Conditioning on extra information does not sacrifice Type 1 error control, and the extra information can be cleverly chosen such that the modified p-value is simple to compute under a multivariate normality assumption. Because scRNA-seq data consists of non-negative integers, a normality assumption is not suitable, and so this approach does not apply. Second, as the conditioning event in (3.7) must be explicitly characterized, each choice of $\widehat{L}(\cdot)$ will require its own bespoke strategy. This is problematic in the scRNA-seq setting, where many specialized

techniques are used for pre-processing, clustering, and trajectory estimation. Zhang et al. [2019] overcome this challenge in the context of clustering by combining cell splitting (Section 3.3.3) with selective inference. The idea is to condition on the test set labeling event $\left\{\widehat{L}\left(X^{\text{train}}, \mathbf{X}^{\text{test}}\right) = \widehat{L}\left(X^{\text{train}}, X^{\text{test}}\right)\right\}$, which can be characterized provided that the estimated clusters are linearly separable. However, their work requires a normality assumption, and does not extend naturally to the setting of trajectory estimation.

Selective inference is omitted as a comparison method in Figure 3.2(a) because we are not aware of a way to compute (3.7) for Poisson data, even for our simple choice for $\widehat{L}(\cdot)$. In Appendix B.3, we show that the selective inference method of Gao et al. [2022], which provides finite-sample valid inference after clustering for a related problem under a normality assumption, does not control the Type 1 error rate when applied to log-transformed Poisson data.

3.3.6 Jackstraw

Much of the difficulty with applying a selective inference approach in the setting of this chapter lies in analytically characterizing the conditioning event in (3.7) and (3.8). An alternative approach is to try to compute a probability similar to (3.8), but without the conditioning event. While the distribution of $\widehat{\beta}_1\left(\widehat{L}(\mathbf{X}), \mathbf{X}_j\right)$ is typically not analytically tractable, its null distribution can be approximated via permutation. This is the idea behind the *jackstraw* method of Chung and Storey [2015], which was originally proposed to test for association between the principal components and features of a data matrix, and was later extended to the clustering setting [Chung, 2020]. To make our discussion of jackstraw congruent with the rest of this chapter, we instantiate the framework to our latent variable GLM setting, and assume

$$\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} H(\Lambda_{ij}), \qquad \log\left(\Lambda_{ij}\right) = \beta_{0j} + \beta_{1j} L_i, \qquad \beta_{1j}, L_i \in \mathbb{R}, \tag{3.9}$$

where $H(\mu)$ is a distribution parameterized by its mean μ. Compared to (4.1), we have omitted the size factors and have assumed that the latent variable is one-dimensional, as these are the assumptions made in the motivating example in Section 3.3.1. We have additionally assumed that each element of X is independently drawn from a distribution $H(\cdot)$.

To test whether the jth gene is differentially expressed, jackstraw creates datasets $X^{\text{permute,b}}$ for $b = 1, \ldots, B$ by randomly permuting the jth column B times. Each dataset gives rise to a GLM coefficient $\widehat{\beta}\left(\widehat{L}(X^{\text{permute,b}}), X_j^{\text{permute,b}}\right)$ and an associated standard error estimate $\widehat{\text{SE}}\left(\widehat{\beta}\left(\widehat{L}(X^{\text{permute,b}}), X_j^{\text{permute,b}}\right)\right)$. The p-value for the jth gene is computed as

$$\frac{1}{B}\sum_{b=1}^{B} \mathbf{1}\left\{\frac{\left|\widehat{\beta}_1\left(\widehat{L}(X^{\text{permute,b}}), X_j^{\text{permute,b}}\right)\right|}{\widehat{\text{SE}}\left(\widehat{\beta}_1\left(\widehat{L}(X^{\text{permute,b}}), X_j^{\text{permute,b}}\right)\right)} \geq \frac{\left|\widehat{\beta}_1\left(\widehat{L}(X), X_j\right)\right|}{\widehat{\text{SE}}\left(\widehat{\beta}_1\left(\widehat{L}(X), X_j\right)\right)}\right\}, \tag{3.10}$$

where $\mathbf{1}\{\cdot\}$ is an indicator function which evaluates to 1 if the inequality is true and 0 otherwise.

Under the null hypothesis that $\beta_{1j} = 0$ in (3.9), $X^{\text{permute,b}}$ and X have the same distribution. Furthermore, both sides of the inequality in (3.10) use the same data realization to construct the predictor and the response. This suggests that under the null hypothesis that $\beta_{1j} = 0$ in (3.9), the distribution of the p-value in (3.10) will converge to Unif$(0, 1)$ as $B \to \infty$. This null hypothesis is slightly different than the ones tested in Sections 3.3.2–3.3.5, which involved association between the jth gene and an *estimated* latent variable.

Unfortunately, carrying out jackstraw as described above is computationally infeasible, as testing $H_0 : \beta_{1j} = 0$ for $j = 1, \ldots, p$ requires $B \times p$ computations of $\widehat{L}(\cdot)$. To improve computational efficiency, Chung and Storey [2015] suggest randomly choosing a set of s genes S_b to permute to obtain $X^{\text{permute,b}}$ for $b = 1, \ldots, B$. Then, the p-value for the jth gene is computed as

$$\frac{1}{B \times s}\sum_{b=1}^{B}\sum_{q \in S_b} \mathbf{1}\left\{\frac{\left|\widehat{\beta}_1\left(\widehat{L}(X^{\text{permute,b}}), X_q^{\text{permute,b}}\right)\right|}{\widehat{\text{SE}}\left(\widehat{\beta}_1\left(\widehat{L}(X^{\text{permute,b}}), X_q^{\text{permute,b}}\right)\right)} \geq \frac{\left|\widehat{\beta}_1\left(\widehat{L}(X), X_j\right)\right|}{\widehat{\text{SE}}\left(\widehat{\beta}_1\left(\widehat{L}(X), X_j\right)\right)}\right\}. \tag{3.11}$$

Since all p genes are compared to the same reference distribution, only B total computations of $\widehat{L}(\cdot)$ are needed. Unfortunately, the p-value in (3.11) only follows a Unif$(0,1)$ distribution if the quantity on the right-hand side of the inequality in (3.11) follows the same distribution for $j = 1, \ldots, p$. This does not hold in the simple example from Section 3.3.1, where the quantity has a different distribution for the genes with $\Lambda_{ij} = 1$ than those with $\Lambda_{ij} = 10$. While the collection of p-values aggregated across all of the genes appears to follow a Unif$(0,1)$ distribution (Figure 3.2(a), left), some genes have anti-conservative p-values (Figure 3.2(a), center) and others have overly conservative p-values (Figure 3.2(a), right).

3.3.7 PseudotimeDE

Song and Li [2021] recently proposed PseudotimeDE to test a gene's association with an estimated trajectory. Here, we present a slight modification of their proposal, which is tailored to our setting but does not change the fundamental properties with respect to the discussion. Implementation details are provided in Appendix B.4. For $b = 1, \ldots, B$, PseudotimeDE subsamples the cells in X to obtain X^b, computes $\widehat{L}(X^b)$, and then permutes this vector to create $\Pi\left(\widehat{L}(X^b)\right)$, where $\Pi(\cdot)$ is a permutation operator. It then computes $\widehat{\beta}\left(\Pi\left(\widehat{L}(X^b)\right), X_j^b\right)$ for each of the B subsamples, for $\hat{\beta}(\cdot,\cdot)$ defined in (3.3). The empirical p-value for the jth gene is given by

$$\frac{1}{B}\sum_{i=1}^{B} \mathbf{1}\left\{\left|\widehat{\beta}\left(\Pi\left(\widehat{L}(X^b)\right), X_j^b\right)\right| \geq \left|\widehat{\beta}\left(\widehat{L}(X), X_j\right)\right|\right\}. \tag{3.12}$$

While it is not entirely clear what null hypothesis PseudotimeDE is designed to test, we can see that there is a problem with the p-value in (3.12). On the right-hand side of the inequality in (3.12), there is association between the predictor and the response in the GLM due to the fact that both are generated from the data X. On the left-hand side of the inequality in (3.12), permuting $\widehat{L}(X^b)$ disrupts the association between the predictor and the response. Thus, even in the absence of any signal in the data, the quantity on the right-hand side of (3.12) does not have the same distribution as the quantity on the left-hand side of (3.12).

As shown in Figure 3.2(a), under our simple example where there is no true trajectory, the p-values from (3.12) are anti-conservative.

3.4 Count splitting

3.4.1 Method

In Sections 3.3.3 and 3.3.4, we saw that cell splitting and gene splitting are not suitable options for latent variable inference on scRNA-seq data. Here, we propose *count splitting*, which involves splitting the expression counts themselves, rather than the genes or the cells, to carry out latent variable estimation and differential expression analysis. The algorithm is as follows.

Algorithm 3.1 (Count splitting for latent variable inference) *For a constant ϵ with $0 < \epsilon < 1$,*

Step 0: Count splitting. *Draw $\mathbf{X}_{ij}^{\text{train}} \mid \{\mathbf{X}_{ij} = X_{ij}\} \overset{\text{ind.}}{\sim} \text{Binomial}\left(X_{ij}, \epsilon\right)$, and let $X^{\text{test}} = X - X^{\text{train}}$.*

Step 1: Latent variable estimation. *Compute $\widehat{L}\left(X^{\text{train}}\right)$.*

Step 2: Differential expression analysis. *For $j = 1, \ldots, p$,*

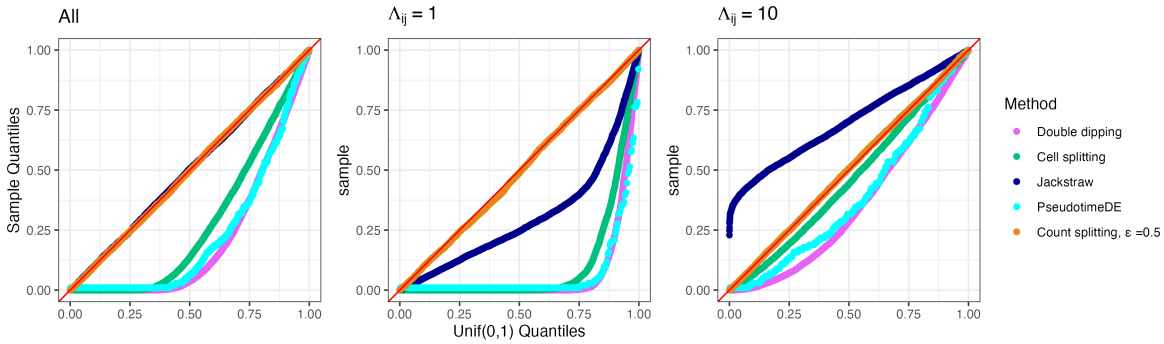
(a) Fit a GLM with a log link to predict X_j^{test} using $\widehat{L}(X^{\text{train}})$, with the γ_i included as offsets. This provides $\widehat{\beta}_1\left(\widehat{L}\left(X^{\text{train}}\right), X_j^{\text{test}}\right)$, an estimate of $\beta_1\left(\widehat{L}\left(X^{\text{train}}\right), \mathbf{X}_j^{\text{test}}\right)$.

(b) Compute a Wald p-value for $H_0 : \beta_1\left(\widehat{L}\left(X^{\text{train}}\right), \mathbf{X}_j^{\text{test}}\right) = 0$ vs. $H_1 : \beta_1\left(\widehat{L}\left(X^{\text{train}}\right), \mathbf{X}_j^{\text{test}}\right) \neq 0$, which takes the form

$$Pr_{H_0:\ \beta_1\left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)=0}\left(\left|\widehat{\beta}_1\left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)\right| \geq \left|\widehat{\beta}_1\left(\widehat{L}\left(X^{\text{train}}\right), X_j^{\text{test}}\right)\right|\right). \quad (3.13)$$

Note that in Step 2(b), we are computing a standard GLM Wald p-value for a regression of X_j^{test} onto $\widehat{L}(X^{\text{train}})$. The next result is a well-known property of the Poisson distribution.

(a)



(b)

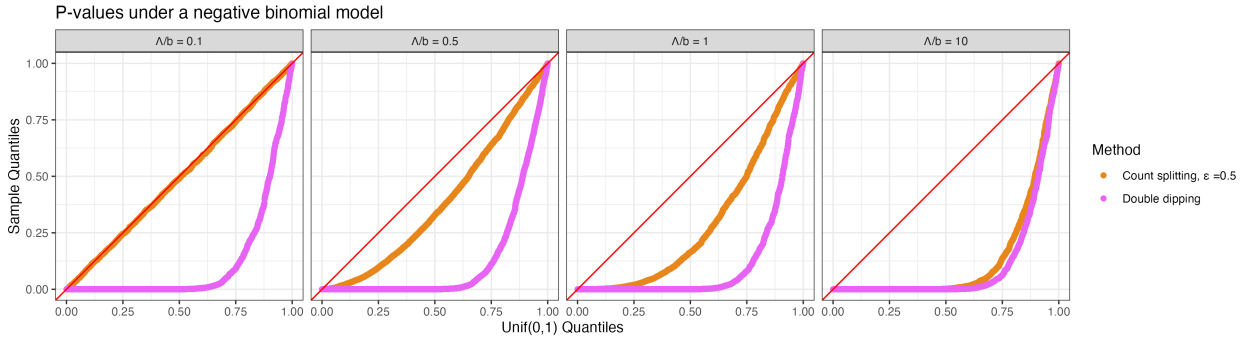


Figure 3.2: Uniform QQ-plots of p-values under the null. *(a):* The data are Poisson. P-values are displayed for the $p = 10$ genes in each of 2000 datasets generated as described in Section 3.3.1. The left-hand panel shows all of the genes aggregated, whereas the center and right panels break the results down by the value of Λ_{ij}. *(b):* The data are negative binomial. P-values are displayed for the $p = 10$ genes in each dataset for the simulation described in Section 3.4.2. Results are broken down by the magnitude of $\frac{\Lambda}{b}$. As $\frac{\Lambda}{b}$ increases, the correlation between $\mathbf{X}^{\text{train}}$ and $\mathbf{X}^{\text{test}}$ increases, and the performance of count splitting approaches that of the double dipping method.

Proposition 3.4.1 (Binomial thinning of Poisson processes) *If* $\mathbf{X}_{ij} \sim Poisson(\gamma_i \Lambda_{ij})$, *then* $\mathbf{X}_{ij}^{\text{train}}$ *and* $\mathbf{X}_{ij}^{\text{test}}$, *as constructed in Algorithm 3.1, are independent. Furthermore,* $\mathbf{X}_{ij}^{\text{train}} \sim Poisson(\epsilon \gamma_i \Lambda_{ij})$ *and* $\mathbf{X}_{ij}^{\text{test}} \sim Poisson((1-\epsilon)\gamma_i \Lambda_{ij})$. *See Durrett [2019], Section 3.7.2, for details.*

This means that in (3.13), under a Poisson model, the predictor and the response are independent on both sides of the inequality. Consequently, the p-value in (3.13) will retain all standard properties of a GLM Wald p-value: e.g. control of the Type 1 error rate for $H_0 : \beta_1 \left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right) = 0$ when n is sufficiently large. Furthermore, when n is sufficiently large, we can invert the test in (3.13) to obtain confidence intervals with $100 \times (1-\alpha)\%$ coverage for the parameter $\beta_1 \left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)$.

We now consider the parameter $\beta_1 \left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)$. Suppose that for any matrix M and scalar a, the function $\widehat{L}(\cdot)$ satisfies $\widehat{L}(aM) \propto \widehat{L}(M)$. In this case, Proposition 3.4.1 says that $\widehat{L}\left(E[\mathbf{X}^{\text{train}}]\right) = \widehat{L}\left(\epsilon E[\mathbf{X}]\right) \propto \widehat{L}\left(E[\mathbf{X}]\right)$. Furthermore, if $\log\left(E[\mathbf{X}_j]\right) = \beta_0 + \beta_1^T L_i$, then $\log\left(E[\mathbf{X}_j^{\text{test}}]\right) = \log(1-\epsilon) + \beta_0 + \beta_1^T L_i$. Therefore, $\beta_1 \left(\widehat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)$ is closely related to $\beta_1 \left(\widehat{L}(X), \mathbf{X}_j\right)$, the parameter (unsuccessfully) targeted by the double dipping method (Section 3.3.2).

Remark 3.1 *The key insight behind Algorithm 3.1 is that, under a Poisson assumption, a test for association between* $\widehat{L}\left(X^{\text{train}}\right)$ *and* $\mathbf{X}_j^{\text{test}}$ *that uses* $\widehat{L}\left(X^{\text{train}}\right)$ *and* X_j^{test} *will inherit standard statistical guarantees, despite the fact that* $\widehat{L}\left(X^{\text{train}}\right)$ *and* X_j^{test} *are functions of the same data. This insight did not rely on the use of a GLM or a Wald test in Step 2 of Algorithm 3.1. Thus, other approaches could be used to quantify this association.*

Figure 3.2(a) shows that count splitting with $\epsilon = 0.5$ produces uniformly distributed p-values for all genes in the example described in Section 3.3.1, using a Poisson GLM in Step 2 of Algorithm 3.1. We explore more values of ϵ and more complicated scenarios in Section 3.5.

Count splitting is a special case of "data fission", proposed in a preprint by Leiner et al. [2022] while this chapter was in preparation. While the data fission framework is broad enough to encompass this latent variable setting, Leiner et al. [2022] focus on comparing data

fission to sample splitting in supervised settings where the latter is an option. In unsupervised settings, where sample splitting is not an option (as seen in Section 3.3.3), ideas similar to count splitting have been applied by Batson et al. [2019] and Chen et al. [2021] for tasks such as evaluating the goodness-of-fit of a low-rank approximation of a matrix. We elaborate on connections to Batson et al. [2019] in Appendix B.2 of the supplementary materials. Finally, Gerard [2020] suggest applying binomial thinning to real scRNA-seq datasets to generate synthetic datasets to use for comparing and evaluating scRNA-seq software packages and methods, and this approach is used in the supplement of Sarkar and Stephens [2021] to compare various scRNA-seq models.

3.4.2 What if the data are not Poisson?

The independence between $\mathbf{X}_{ij}^{\text{train}}$ and $\mathbf{X}_{ij}^{\text{test}}$ in Proposition 3.4.1 holds if and only if $\mathbf{X}_{ij}$ has a Poisson distribution [Kimeldorf et al., 1981]. Thus, if we apply Algorithm 3.1 to data that are not Poisson, there will be dependence between the predictor and the response on the right-hand side of the inequality in (3.13), and the p-value in (3.13) will not be uniformly distributed under H_0.

Wang et al. [2018] argue that the Poisson model is sufficient to model scRNA-seq data. Townes et al. [2019] advocate modeling the counts for each cell with a multinomial distribution. When the number of genes is large, the elements of the multinomial can be well-approximated by independent Poisson distributions, and indeed the authors rely on this approximation for computational reasons. Similarly, Batson et al. [2019] assume that $\mathbf{X}_{ij} \sim \text{Binomial}(\Omega_{ij}, p_i)$, but note that typically Ω_{ij} is large and p_i is small, so that a Poisson approximation applies.

Sarkar and Stephens [2021] advocate pairing a Poisson *measurement* model for scRNA-seq data with a separate *expression* model to account for overdispersion compared to the Poisson model. For example, a Gamma expression model leads to

$$\mathbf{X}_{ij} \mid \{\boldsymbol{\tau}_{ij} = \tau_{ij}\} \sim \text{Poisson}\left(\Lambda_{ij}\tau_{ij}\right), \quad \boldsymbol{\tau}_{ij} \sim \text{Gamma}(b_j, b_j), \tag{3.14}$$

where we have omitted size factors for simplicity. This induces a negative binomial marginal distribution on $\mathbf{X}_{ij}$, where $\mathrm{E}[\mathbf{X}_{ij}] = \Lambda_{ij}$ and $\mathrm{Var}(\mathbf{X}_{ij}) = \Lambda_{ij} + \frac{\Lambda_{ij}^2}{b_j}$. If $\mathbf{X}_{ij}$ is drawn from (3.14) and Algorithm 3.1 is applied, then $\mathbf{X}_{ij}^{\text{train}}$ and $\mathbf{X}_{ij}^{\text{test}}$ each follow negative binomial distributions, with $\mathrm{E}[\mathbf{X}_{ij}^{\text{train}}] = \epsilon\,\mathrm{E}[\mathbf{X}_{ij}]$ and $\mathrm{E}[\mathbf{X}^{\text{test}}] = (1-\epsilon)\,\mathrm{E}[\mathbf{X}]$ [Harremoës et al., 2010]. Our next result, proven in Appendix B.5.1, quantifies the correlation between $\mathbf{X}_{ij}^{\text{train}}$ and $\mathbf{X}_{ij}^{\text{test}}$ in this setting.

Proposition 3.4.2 *Suppose that $\mathbf{X}_{ij}$ follows a negative binomial distribution with expected value Λ_{ij} and variance $\Lambda_{ij} + \frac{\Lambda_{ij}^2}{b_j}$. If we perform Step 0 of Algorithm 3.1, then*

$$\mathrm{Cor}\left(\mathbf{X}_{ij}^{\text{train}}, \mathbf{X}_{ij}^{\text{test}}\right) = \frac{\sqrt{\epsilon(1-\epsilon)}}{\sqrt{\epsilon(1-\epsilon) + \frac{b_j^2}{\Lambda_{ij}^2} + \frac{b_j}{\Lambda_{ij}}}}. \tag{3.15}$$

To investigate the performance of count splitting under overdispersion, we generate datasets under (3.14) with $n = 200$ and $p = 10$. For each dataset, $\Lambda_{ij} = \Lambda = 5$ for $i = 1, \ldots, n$ and $j = 1, \ldots, p$, and $b_j = b$ for $j = 1, \ldots, p$, so that every element of X is drawn from the same distribution. We generate 500 datasets for each value of $b \in \{50, 10, 5, 0.5\}$, so that $\frac{\Lambda}{b} \in \{0.1, 0.5, 1, 10\}$. Figure 3.2(b) displays the Wald p-values that result from running Algorithm 3.1 with a negative binomial GLM in Step 2, for all genes across all of the simulated datasets.

The denominator of (3.15) in Proposition 3.4.2 shows that $\frac{\Lambda}{b}$ determines the extent of correlation between $\mathbf{X}^{\text{train}}$ and $\mathbf{X}^{\text{test}}$ in this setting. As shown in Figure 3.2(b), when $\frac{\Lambda}{b}$ is small, count splitting produces approximately uniformly distributed p-values, and thus comes very close to controlling the Type 1 error rate. As $\frac{\Lambda}{b}$ grows, the performance of count splitting approaches that of the double dipping method discussed in Section 3.3.2. Thus, count splitting tends to outperform the double dipping method, and in the case of extremely high overdispersion will be *no worse than* the double dipping method. For the real scRNA-seq dataset considered in Section 3.6, we show in Appendix B.6 that the majority of the estimated values of $\frac{\Lambda_{ij}}{b_j}$ are less than 1.

3.4.3 Choosing the tuning parameter ϵ

The parameter ϵ in Algorithm 3.1 governs a tradeoff between the information available for estimating L and the information available for carrying out inference. Proposition 3.4.3, proven in Appendix B.5.2, formalizes the intuition that X^{train} will look more similar to X when ϵ is large.

Proposition 3.4.3 *If* $\mathbf{X}_{ij} \sim \text{Poisson}(\gamma_i \Lambda_{ij})$, *then* $\text{Cor}(\mathbf{X}_{ij}, \mathbf{X}_{ij}^{\text{train}}) = \sqrt{\epsilon}$.

Thus, as ϵ decreases, we expect $\widehat{L}(X^{\text{train}})$ and $\widehat{L}(X)$ to look less similar. This is a drawback, as scientists would ideally like to estimate L using *all* of the data. However, as ϵ increases, the power to reject false null hypotheses in Step 2(b) of Algorithm 3.1 decreases. Proposition 3.4.4, proven in Appendix B.5.3, quantifies this loss of power.

Proposition 3.4.4 *Let* $\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} \text{Poisson}\left(\gamma_i \exp(\beta_{0j} + \beta_{1j} L_i)\right)$. *Then* $Var\left(\widehat{\beta}_1(L, \mathbf{X}_j^{\text{test}})\right) \approx \frac{1}{1-\epsilon} Var\left(\widehat{\beta}_1(L, \mathbf{X}_j)\right)$.

In the ideal setting where $\widehat{L}(X^{\text{train}}) = L$ and $L \in \mathbb{R}^{n \times 1}$ for simplicity, using $\mathbf{X}_j^{\text{test}}$ rather than $\mathbf{X}_j$ as the response inflates the variance of the estimated coefficient by a factor of $\frac{1}{1-\epsilon}$. Thus, when ϵ is large, Step 2(b) of Algorithm 3.1 has lower power. In practice, we recommend setting $\epsilon = 0.5$ to balance the dual goals of latent variable estimation and downstream inference.

3.5 Simulation study

3.5.1 Data generating mechanism

We generate data from (4.1) and (3.4) with $n = 2700$ and $p = 2000$. We generate the size factors $\gamma_i \overset{\text{ind.}}{\sim} \text{Gamma}\,(10, 10)$ and treat them as known. Throughout this section, whenever we perform count splitting, we fit a Poisson GLM in Step 2 of Algorithm 3.1 and report Wald p-values.

In this section, we investigate the performance of count splitting under two data-generating mechanisms: one generates a true continuous trajectory and the other generates true clusters. To generate data with an underlying trajectory, we set $L = \left(I_n - \frac{1}{n}11^T\right) Z$, where $Z_i \overset{\text{ind.}}{\sim} N(0,1)$. Under (4.1), L is the first principal component of the matrix $\log(\Lambda)$. To estimate L, we take the first principal component of the matrix $\log(\text{diag}(\gamma)^{-1}X + 11^T)$. To generate data with underlying cell types, we let $L_i \overset{\text{ind.}}{\sim} \text{Bernoulli}(0.5)$ in (4.1), indicating membership in one of two cell types. We estimate L by running k-means with $k = 2$ on the matrix $\log\left(\text{diag}(\gamma)^{-1}X + 11^T\right)$. (In each case, we have used a pseudocount of one to avoid taking the log of zero.)

In our primary simulation setting, the value β_{0j} for each gene is randomly chosen to be either $\log(3)$ or $\log(25)$ with equal probability, such that the data includes a mix of low-intercept and high-intercept genes. For each dataset, we let $\beta_{1j} = 0$ for 90% of the p genes. Under (4.1), $\Lambda_{1j} = \ldots = \Lambda_{nj}$ for these genes and thus $\beta_1(Z, \mathbf{X}_j) = 0$ for any estimated latent variable Z. Thus, we refer to these as the null genes. The remaining 10% of the genes have the same non-zero value of β_{1j}; these are the differentially-expressed genes. For each latent variable setting, we generate 100 datasets for each of 15 equally-spaced values of β_{1j} in $[0.18, 3]$.

3.5.2 Type 1 error results

Figure 3.3(a) shows that count splitting controls the Type 1 error rate for a range of ϵ values for the 90% of genes for which $\beta_{1j} = 0$ in datasets that are generated as specified in Section 3.5.1.

3.5.3 Quality of estimate of L

As mentioned in Section 3.4.3, smaller values of ϵ in count splitting compromise our ability to accurately estimate the unobserved latent variable L. To quantify the quality of our estimate of L, in the trajectory estimation case we compute the absolute value of the correlation between L and $\widehat{L}\left(X^{\text{train}}\right)$, and in the clustering setting we compute the adjusted Rand index

[Hubert and Arabie, 1985] between L and $\widehat{L}\left(X^{\text{train}}\right)$. The results are shown in Figure 3.3(b). Here, we consider three settings for the value of β_{0j}: (i) each gene is equally likely to have $\beta_{0j} = \log(3)$ or $\beta_{0j} = \log(25)$ (as in Section 3.5.2); (ii) all genes have $\beta_{0j} = \log(3)$ ("100% low-intercept" setting); and (iii) all genes have $\beta_{0j} = \log(25)$ ("100% high-intercept" setting). The 100% low-intercept setting represents a case where the sequencing was less deep, and thus the data more sparse.

After taking a log transformation, the genes with $\beta_{0j} = \log(3)$ have higher variance than those with $\beta_{0j} = \log(25)$. Thus, there is more noise in the "100% low-intercept" setting than in the "100% high-intercept" setting. As a result, for a given value of ϵ, the "100% low-intercept" setting results in the lowest-quality estimate of L. Estimation of L is particularly poor in the "100% low-intercept" setting when ϵ is small, since then X^{train} contains many zero counts. This suggests that count splitting, especially with small values of ϵ, is less effective on shallow sequencing data.

3.5.4 Power results

For each dataset and each gene, we compute the true parameter $\beta_1\left(\widehat{L}\left(X^{\text{train}}\right), \mathbf{X}_j^{\text{test}}\right)$ defined in (3.2). For a Poisson GLM (i.e. $p_\theta(\cdot)$ in (3.2) is the density of a Poisson(θ) distribution), it is straightforward to show that we can compute $\beta_1\left(\widehat{L}\left(X^{\text{train}}\right), \mathbf{X}_j^{\text{test}}\right)$ by fitting a Poisson GLM with a log link to predict $E\left[\mathbf{X}_j\right]$ using $\widehat{L}\left(X^{\text{train}}\right)$, with the γ_i included as offsets.

Figure 3.3(c) shows that our ability to reject the null hypothesis depends on this true parameter, as well as on the value of ϵ. These results are shown in the setting where 50% of the genes have intercept log(3) and the other 50% have intercept log(25). The impact of ϵ on power is more apparent for genes with smaller intercepts, as these genes have even less information left over for inference when $(1-\epsilon)$ is small. This result again suggests that count splitting will work best on deeply sequenced data where expression counts are less sparse.

As suggested by Section 3.4.3, Figures 3.3(a) and 3.3(c) show a tradeoff in choosing ϵ: a larger value of ϵ improves latent variable estimation, but yields lower power for differential expression analysis. In practice, we recommend choosing $\epsilon = 0.5$.

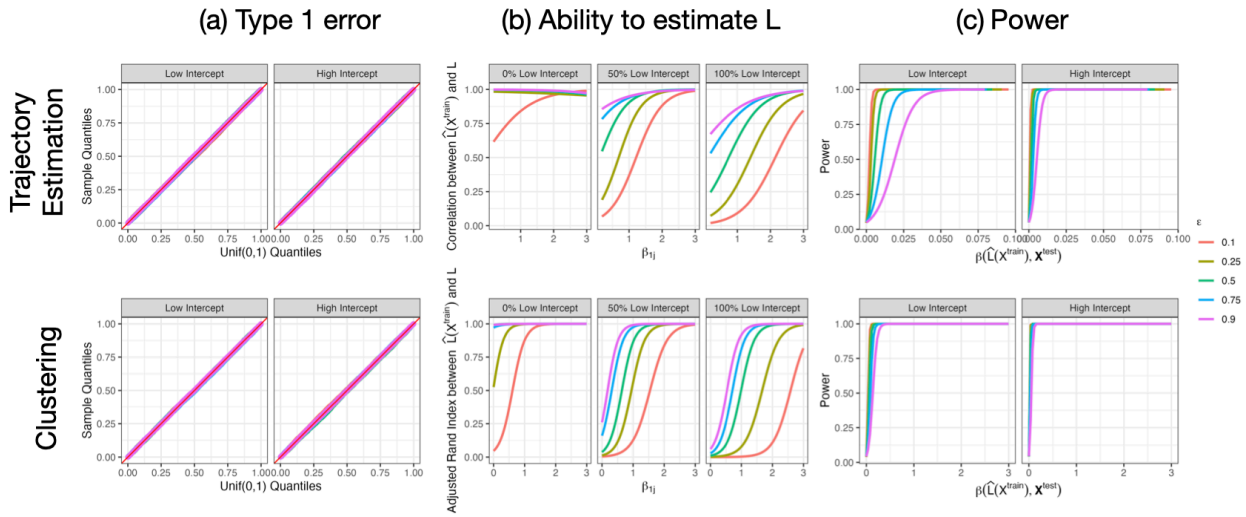


Figure 3.3: *(a)* Uniform QQ plots of GLM p-values for all null genes from the simulations described in Section 3.5.1. *(b)* Quality of estimate of L, as defined in Section 3.5.3, as a function of β_{1j} and the percent of low-intercept genes in the dataset. *(c)* The proportion of null hypotheses that are rejected, aggregated across all non-null genes for all datasets generated with 50% low-intercept and 50% high-intercept genes. The parameter $\beta_1\left(\hat{L}(X^{\text{train}}), \mathbf{X}_j^{\text{test}}\right)$ is defined in (3.2).

3.5.5 Coverage results

For each dataset and each gene, we compute a 95% Wald confidence interval for the slope parameter in the GLM. As shown in Table 3.1, these intervals achieve nominal coverage for the target parameter $\beta_1\left(\widehat{L}\left(X^{\text{train}}\right), \mathbf{X}_j^{\text{test}}\right)$, defined in (3.2) and computed as in Section 3.5.4.

	Trajectory Estimation		Clustering	
	Low Intercept	High Intercept	Low Intercept	High Intercept
Null genes	0.950	0.950	0.949	0.950
Non-null genes	0.955	0.956	0.949	0.950

Table 3.1: The proportion of 95% confidence intervals that contain $\beta_1\left(\widehat{L}\left(X^{\text{train}}\right), \mathbf{X}_j^{\text{test}}\right)$, aggregated across genes and across values of ϵ.

3.6 Application to cardiomyocyte differentiation data

Elorbany et al. [2022] collect single-cell RNA-sequencing data at seven unique time points (over 15 days) in 19 human cell lines. The cells began as induced pluripotent stem cells (IPSCs) on day 0, and over the course of 15 days they differentiated along a bifurcating trajectory into either cardiomyocytes (CMs) or cardiac fibroblasts (CFs). We are interested in studying genes that are differentially expressed along the trajectory from IPSC to CM. Throughout this analysis, we ignore the true temporal information (the known day of collection).

Starting with the entire dataset X, we first perform count splitting with $\epsilon = 0.5$ to obtain X^{train} and X^{test}. We then perform the lineage estimation task from Elorbany et al. [2022] on X^{train} to come up with a subset of 10,000 cells estimated to lie on the IPSC to CM trajectory. We retain only these 10,000 cells for further analysis. In what follows, to facilitate comparison between the double dipping method and our count splitting approach, we will use this same subset of 10,000 cells for all methods, even though in practice the double dipping method would have chosen the lineage subset based on *all* of the data, rather than on X^{train} alone.

Next, we estimate a continuous differentiation trajectory using the `orderCells()` function from the `Monocle3` package in `R` [Pliner et al., 2022]. Details are given in Appendix B.7. To estimate the size factors $\gamma_1, \ldots, \gamma_n$ in (3.4), we let $\hat{\gamma}_i(X)$ be the normalized row sums of the expression matrix X, as is the default in the `Monocle3` package. Throughout this section, for $j = 1, \ldots, p$, we compare three methods:

Full double dipping: Fit a Poisson GLM of X_j on $\widehat{L}(X)$ with $\hat{\gamma}_i(X)$ included as offsets.

Count splitting: Fit a Poisson GLM of X_j^{test} on $\widehat{L}(X^{\text{train}})$ with $\hat{\gamma}_i(X^{\text{train}})$ included as offsets.

Test double dipping: Fit a Poisson GLM of X_j^{test} on $\widehat{L}(X^{\text{test}})$ with $\hat{\gamma}_i(X^{\text{test}})$ included as offsets.

For each method, we report the Wald p-values for the slope coefficients. The test double dipping method is included to facilitate understanding of the results from the other two methods. We show in Appendix B.6 that the estimated overdispersion parameters are relatively small for this dataset, justifying the use of count splitting and Poisson GLMs in each of the methods above. In conducting count splitting, we ensure that none of the pre-processing steps required to compute $\widehat{L}(X^{\text{train}})$ make use of X^{test}.

We first analyze all $10,000$ cells. There is a true differentiation trajectory in this dataset: cells were sampled at at seven different time points of a directed differentiation protocol, and changes in gene expression as well as phenotype that are characteristic of this differentiation process were observed [Elorbany et al., 2022]. We focus on a subset of $p = 2,500$ high-variance genes that were selected from more than $32,000$ genes that were expressed in the raw data (see Appendix B.7 for details). The left panel of Figure 3.4 displays the count splitting p-values for differential expression of these 2,500 genes against the double dipped p-values.

We first note the general agreement between the three methods: genes that have small p-values with one method generally have small p-values with all the methods. Thus, count splitting tends to identify the same differentially expressed genes as the methods that double dip when there is a true trajectory in the data. That said, we do notice that the full double dipping method tends to give smaller p-values than count splitting. There are two possible reasons for this: (i) the double dipped p-values may be artificially small due to the double dipping, or (ii) it might be due to the fact that the full double dipping method uses twice as much data. In this setting, it seems that (ii) is the correct explanation: the test double dipping method (which uses the same amount of data as count splitting both to estimate L and to test the null hypothesis) does not yield smaller p-values than count splitting. It seems that there is enough true signal in this data that most of the genes identified by the double dipped methods as differentially expressed are truly differentially expressed, rather than false positives attributable to double dipping.

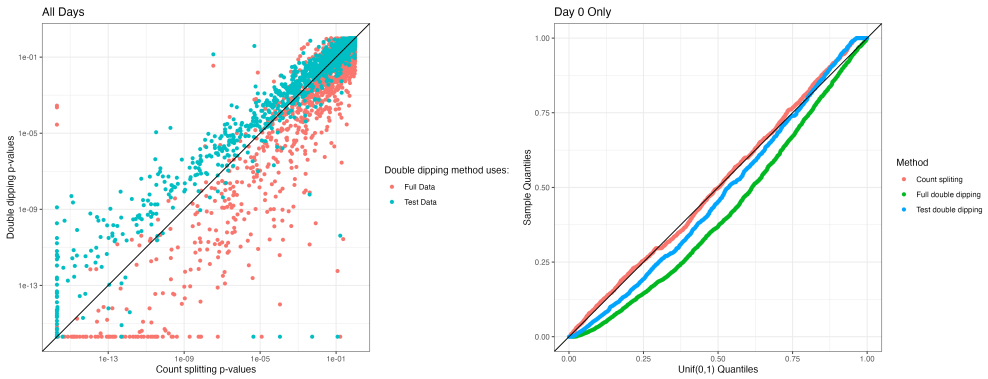


Figure 3.4: *Left:* Count splitting p-values vs. double dipped p-values for $2,500$ genes, on a log scale, when all $10,000$ cells are used in the analysis. *Right:* Uniform QQ plot of p-values for $2,500$ genes obtained from each method when only the Day 0 cells are used.

Next, we subset the 10,000 cells to only include the $2,303$ cells that were measured at Day 0 of the experiment, before differentiation had begun. Then, despite knowing that these cells should be largely homogeneous, we apply $\widehat{L}(\cdot)$ to estimate pseudotime. We note that the $\widehat{L}(\cdot)$ function described in Appendix B.7 controls for cell cycle-related variation in gene expression so that this signal cannot be mistaken for pseudotime. We suspect that in this setting, any association seen between pseudotime and the genes is due to overfitting or random noise. We select a new set of $2,500$ high variance genes using this subset of cells. The right panel of Figure 3.4 shows uniform QQ plots of the p-values for these $2,500$ genes from each of the three methods. In the absence of real differentiation signal, the p-values should follow a uniform distribution. We see that count splitting controls the Type 1 error rate, while the methods that double dip do not. Unlike the previous example that had a very strong true differentiation signal, even test double dipping yields false positives. Without assessment, the level of true signal in any dataset cannot be assumed to outweigh the effects of double dipping.

In summary, count splitting detects differentially expressed genes when there is true signal in the data, and protects against Type 1 errors when there is no true signal in the data.

3.7 Discussion

Under a Poisson assumption, count splitting provides a flexible framework for carrying out valid inference after latent variable estimation that can be applied to virtually any latent variable estimation method and inference technique. This has important applications in the growing field of pseudotime or trajectory analysis, as well as in cell type analysis.

We expect count splitting to be useful in situations other than those considered in this chapter. For example, as explored in Appendix B.3, count splitting can be used to test the overall difference in means between two estimated clusters, as considered by Gao et al. [2022] and Chen and Witten [2023]. As suggested in related work by Batson et al. [2019], Chen et al. [2021], and Gerard [2020], count splitting could be used for model selection tasks such as choosing how many dimensions to keep when reducing the dimension of Poisson scRNA-seq data. Finally, count splitting may lead to power improvements over sample splitting for Poisson data on tasks where the latter is an option [Leiner et al., 2022].

In this chapter, we assume that after accounting for heterogenous expected expression across cells and genes, the scRNA-seq data follows a Poisson distribution. Some authors have argued that scRNA-seq data is overdispersed. As discussed in Section 3.4.2, count splitting fails to provide independent training and testing sets when applied to negative binomial data. **?** suggest that inference can be carried out in this setting by working with the conditional distribution of $X^{\text{test}} \mid X^{\text{train}}$. Unfortunately, this conditional distribution does not lend itself to inference on parameters of interest. In Chapter 4, we extend count splitting to the negative binomial distribution.

Code for reproducing the simulations and real data analysis in this chapter is available at `github.com/anna-neufeld/countsplit_paper`. An `R` package with tutorials is available at `anna-neufeld.github.io/countsplit.tutorials`.

Chapter 4

NEGATIVE BINOMIAL COUNT SPLITTING FOR SINGLE-CELL RNA SEQUENCING DATA

The contents of this chapter are included in Neufeld et al. [2023b] (to be submitted).

4.1 Introduction

A single-cell RNA sequencing (scRNA-seq) dataset involving n cells and p genes can be written as a matrix $X \in \mathbb{Z}_{\geq 0}^{n \times p}$, where entry X_{ij} is the number of reads or unique molecular identifiers from the ith cell that map to the jth gene. It is common to assume that X is a realization from a random variable $\mathbf{X}$, and that

$$E\left[\mathbf{X}_{ij}\right] = \gamma_i \Lambda_{ij}, \text{ with } g(\Lambda) = L\beta^\top \text{ for } L \in \mathbb{R}^{n \times K}, \beta \in \mathbb{R}^{p \times K}, \tag{4.1}$$

for some link function $g(\cdot)$ [Sarkar and Stephens, 2021]. In (4.1), $\gamma = (\gamma_1, \ldots, \gamma_n)^T$ stores cell-specific *size factors*, which reflect technical variation in sequencing depth between cells, whereas Λ represents the biological variation of interest, which, after applying some suitable link function $g(\cdot)$, is assumed to have rank K for some $K \leq \min(n, p)$.

Fitting the model (4.1) – that is, obtaining estimates $\hat{L}(X)$ and $\hat{\beta}(X)$ of L and β – may be of interest for a number of reasons. For example, we may wish to denoise the data by replacing $\text{diag}(\gamma)^{-1}X$ with $g^{-1}\left(\hat{L}(X)\hat{\beta}(X)^T\right)$ [Eraslan et al., 2019, Lopez et al., 2018, Townes et al., 2019], or we may wish to interpret $\hat{L}(X)$ as a measure of an unobserved aspect of cell state, e.g. cell type or position along a developmental trajectory [Aizarani et al., 2019, Grün et al., 2015, Zhang et al., 2019].

After fitting the model (4.1), we typically want to perform some type of model validation or inference. Here, we give three examples.

Example 4.1 *We want to assess the quality of our low-rank approximation* $\text{diag}(\gamma)^{-1}X \approx g^{-1}\left(\hat{L}(X)\hat{\beta}(X)^T\right)$ *[Sarkar and Stephens, 2021, Batson et al., 2019].*

Example 4.2 *We want to identify genes that are associated with* $\hat{L}(X)$ *[Aizarani et al., 2019, Grün et al., 2015, Van den Berge et al., 2020, Hafemeister and Satija, 2019].*

Example 4.3 *We want to know if our estimated latent variables* $\hat{L}(X)$ *are stable, i.e. would be reproducible on a new, independent realization of* $\mathbf{X}$ *[Cao et al., 2020, Lange et al., 2004, Ullmann et al., 2022].*

It is challenging to perform model validation or inference after fitting model (4.1). In Example 4.1, *because we estimated* L *and* β *on the data* X*, we cannot re-use* X *to assess model fit* [Hastie et al., 2009]. In Example 4.2, *because we estimated* L *on the data* X*, we cannot re-use* X *to test for association* [Taylor and Tibshirani, 2015]. In Example 4.3, *we only have access to one dataset*, so it is unclear how to proceed. While very specialized approaches are available to overcome these challenges in specific instantiations of Example 4.1 [Fu and Perry, 2020, Owen and Perry, 2009, Grabski et al., 2022], Example 4.2 [Gao et al., 2022, Chen et al., 2021, Zhang et al., 2019, Chung and Storey, 2015], and Example 4.3 [Tibshirani and Walther, 2005, Lange et al., 2004], in this chapter we will provide a much more flexible framework for model validation or inference after fitting (4.1), which will be applicable to all three examples.

To illustrate the problem, we generate a toy data matrix $X \in \mathbb{Z}_{\geq 0}^{100\times 2}$ such that each $\mathbf{X}_{ij}$ is drawn independently from a negative binomial distribution with mean 5 and variance 10. This is a special case of (4.1) with $\gamma_1 = \ldots = \gamma_n = 1$, $K = 1$, $L = \mathbf{1}_{100}$, and $\beta = [5, 5]^T$. The data are shown in Figure 4.2(a). To illustrate Example 4.1, we apply k-means clustering to the data for a range of values of k. Though there is one true cluster in this example (all 100 cells are homogenous), the mean squared error (MSE: defined in (4.2) in Section 4) computed on the same data used for clustering is monotone decreasing in k (Figure 4.2(c)), incorrectly suggesting that a larger value of k always leads to a better fit. To illustrate

Example 4.2, we fit a negative binomial generalized linear model (GLM) to test whether the expected expression of gene 1 is associated with the cluster labels when there are $k = 2$ estimated clusters. When we perform this test on the same data used for clustering, we obtain p-values that are much smaller than the Unif$(0, 1)$ distribution (Figure 4.2(d)), and thus do not control the Type 1 error rate (recall that no true clusters are present, and thus there is no true association between the clusters and the features).

The solution here might seem obvious: to split our 100 cells into a training set, used to fit (4.1), and a test set, used for model validation. Unfortunately, this *sample splitting* approach does not work. The issue is that fitting (4.1) using the cells in the training set yields latent variable coordinates for cells in the training set only. To use the test set for validation or inference, we must obtain latent variable coordinates of the cells in the test set. This step involves using the test set data itself, which invalidates downstream evaluation or inference. In the case of our toy data analysis in Figure 4.2, we apply k-means clustering to the cells in the training set, and then assign cluster labels to the cells in the test set using 3-nearest neighbor classification. We see in Figure 1(c) that, over 1000 simulated datasets, the within-cluster MSE computed on the test set (see Appendix C.1 for details) decreases monotonically with the number of clusters, because we used the test set both to compute latent variable coordinates for the cells in the test set and to compute the within-cluster MSE. Similarly, Figure 4.2(d) shows that, over 1,000 simulated datasets, the p-values obtained by using a negative binomial GLM to regress the first gene from the test set onto the test set cluster assignments do not control the Type 1 error rate. We refer the reader to Bro et al. [2008], Fu and Perry [2020], and Owen and Perry [2009] for more discussion of the inadequacy of sample splitting in the setting of Example 4.1, and Gao et al. [2022], Chen and Witten [2023], and Neufeld et al. [2022a] for a related discussion in the setting of Example 4.2.

To illustrate the challenges associated with Example 4.3, we suppose that we estimate $k = 5$ cell types via clustering on the toy dataset in Figure 4.2, and then we wish to see if these clusters are stable. Faced with this task in the analysis of real scRNA-seq data, Cao et al. [2020] use a method that they call "intradataset cross-validation," which involves

treating the cell types estimated via clustering as fixed, and then performing 5-fold cross-validation using a classifier (see Algorithm 4.5). A high degree of agreement between the cell types estimated via clustering and those predicted by the classifier during cross-validation is treated as evidence that the clusters are reproducible. Figure 4.2(e) shows a confusion matrix comparing the cell types estimated via clustering to those predicted using cross-validation (with 5-folds and a support vector machine (SVM) classifier) for the cells in the toy dataset from Figure 4.2(a). Despite the fact that all cells are homogenous in this dataset (and thus the estimated clusters are determined by random noise), 95% of the cells fall on the diagonal of the confusion matrix, falsely suggesting stability of the clusters. The issue is that, since all of the data from all of the cells was used for the clustering step, any downstream model evaluation is compromised, even if the downstream task makes use of cross-validation.

Now, suppose that we were able to sequence the same set of cells twice to obtain two independent datasets $X^{(\text{train})} \in \mathbb{Z}_{\geq 0}^{n \times p}$ and $X^{(\text{test})} \in \mathbb{Z}_{\geq 0}^{n \times p}$ generated from $\mathbf{X}$ in (4.1) (with the same true underlying L and β matrices). We could estimate L and/or β using only $X^{(\text{train})}$, and could then validate the results or conduct inference using $X^{(\text{test})}$. Thus, the challenges associated with Examples 1 and 2 displayed in Figure 1(c) and 1(d) would be entirely avoided. Similarly, we could estimate one set of clusters on $X^{(\text{train})}$ and another set of clusters on $X^{(\text{test})}$ and compare the two clusterings using a metric such as the adjusted Rand Index [Hubert and Arabie, 1985], entirely avoiding the challenge of Example 3.

In practice, we cannot sequence the same set of cells twice. Instead, we propose to use our single dataset X to reverse engineer two datasets $X^{(\text{train})}$ and $X^{(\text{test})}$ that function like two independent sequencing experiments performed on the same sets of genes and cells. Our proposal is an extension of the ideas of Batson et al. [2019], Sarkar and Stephens [2021], and Neufeld et al. [2022a], who perform this reverse engineering under the assumption that $X_{ij} \overset{\text{ind.}}{\sim} \text{Binomial}(\Omega_{ij}, p_i)$ or $X_{ij} \overset{\text{ind.}}{\sim} \text{Poisson}(\Lambda_{ij})$, using the well-known binomial thinning property of a Poisson random variable [Durrett, 2019]. However, in practice, scRNA-seq data are typically overdispersed relative to the Poisson distribution, and so the *Poisson count splitting* procedure developed in Chapter 3 will fail to produce independent training

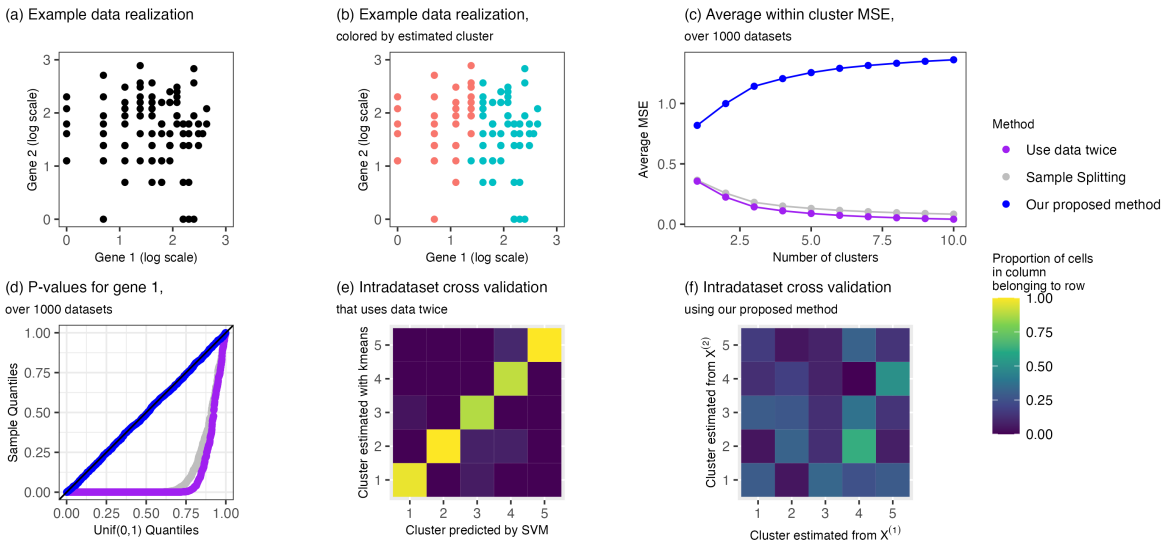


Figure 4.1: *(a)* Data $X \in \mathbb{Z}_{\geq 0}^{100\times 2}$, where each entry X_{ij} is drawn independently from a negative binomial distribution with mean 5 and variance 10. *(b)* The same data X, colored by the clusters estimated when k-means with k=2 is applied to $\log(X+1)$. *(c)* The within-cluster MSE computed after fitting k-means with $k = 1, 2, \ldots, 10$, averaged over 1000 datasets. We compute the MSE using all of the data (purple), we cluster using 50 observations and compute the MSE on the other 50 (gray), and we apply our proposed negative binomial count splitting method (blue). Details are given in Section 4.4.2. *(d)* Uniform QQ-plot of negative binomial GLM p-values for testing for differential expression of gene 1 across the estimated clusters for $1,000$ realizations of X. We cluster and fit the GLM using all of the data (purple), cluster using 50 observations and fit the GLM on the other 50 (gray), and we apply our proposed negative binomial count splitting method (blue). Details are given in Section 4.4.3. *(e)* Confusion matrix resulting from the intradataset cross-validation procedure of Cao et al. [2020] (see Algorithm 4.5 in Section 4.5) that uses the same data for both clustering and validation. *(f)* Confusion matrix resulting from our modified version of intradataset cross-validation (see Algorithm 4.6 in Section 4.5).

and test sets. In this chapter, we introduce *negative binomial count splitting*, which performs this reverse engineering under the assumption that each X_{ij} is drawn independently from a negative binomial distribution.

Figure 4.2(c) and Figure 4.2(d) show that negative binomial count splitting correctly determines that $K = 1$, and controls the Type 1 error rate, in our toy example. We see in Figure 4.2(f) that, after negative binomial count splitting, a confusion matrix of clusterings estimated on $X^{(\text{train})}$ vs. those estimated on $X^{(\text{test})}$ is not diagonal, which correctly shows that the clusters are driven by random noise and are not reproducible on an independent dataset.

Figure 4.2(c) and Figure 4.2(d) show that negative binomial count splitting correctly determines that $K = 1$, and controls the Type 1 error rate, in our toy example. We see in Figure 4.2(f) that, after negative binomial count splitting, a confusion matrix of clusterings estimated on $X^{(\text{train})}$ vs. those estimated on $X^{(\text{test})}$ is not diagonal, which accurately reflects the absence of signal in the toy dataset.

Negative binomial count splitting requires a negative binomial assumption to ensure independence between the training and test sets. However, once the data has been split, we are free to use any latent variable estimation method or inferential technique, including those that do not make use of a negative binomial assumption. So, for instance, one could apply negative binomial count splitting to obtain independent training and test sets, and then fit Poisson generalized linear models.

In Section 4.2, we review the Poisson count splitting procedure of Neufeld et al. [2022a]. In Section 4.3, we introduce negative binomial count splitting and provide some intuition and theoretical results. In Section 4.4, we apply negative binomial count splitting to Example 4.1 and Example 4.2 on simulated data. In Section 4.5, we revisit the intradataset cross-validation procedure of Cao et al. [2020], and re-analyze the stability of the kidney cell types and subtypes from their human cell atlas using negative binomial count splitting.

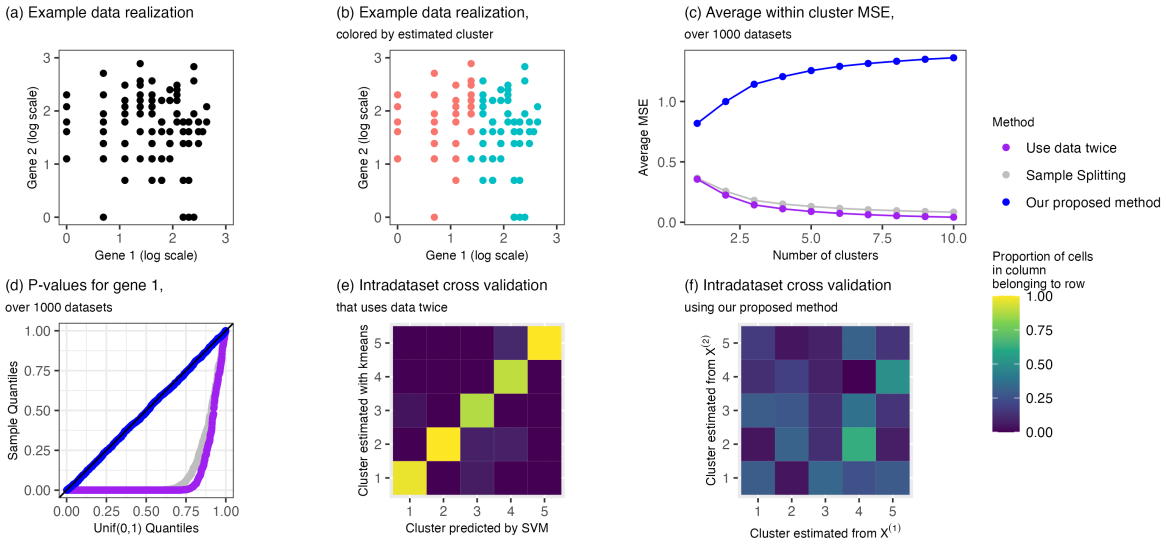


Figure 4.2: *(a)* Data $X \in \mathbb{Z}_{\geq 0}^{100\times 2}$, where each entry X_{ij} is drawn independently from a negative binomial distribution with mean 5 and variance 10. *(b)* The same data X, colored by the clusters estimated when k-means with k=2 is applied to $\log(X+1)$. *(c)* The within-cluster MSE computed after fitting k-means with $k = 1, 2, \ldots, 10$, averaged over 1000 datasets. We compute the MSE using all of the data (purple), we cluster using 50 observations and compute the MSE on the other 50 (gray), and we apply our proposed negative binomial count splitting method (blue). Details are given in Section 4.4.2. *(d)* Uniform QQ-plot of negative binomial GLM p-values for testing for differential expression of gene 1 across the estimated clusters for $1,000$ realizations of X. We cluster and fit the GLM using all of the data (purple), cluster using 50 observations and fit the GLM on the other 50 (gray), and we apply our proposed negative binomial count splitting method (blue). Details are given in Section 4.4.3. *(e)* Confusion matrix resulting from the intradataset cross validation procedure of Cao et al. [2020] (see Algorithm 4.5 in Section 4.5) that uses the same data for both clustering and validation. *(f)* Confusion matrix resulting from our proposed procedure for assessing cluster stability (see Algorithm 4.6 in Section 4.5)

.

4.2 Background

4.2.1 A review of Poisson count splitting

Our goal is to decompose an scRNA-seq dataset $X \in \mathbb{Z}_{\geq 0}^{n \times p}$ into two independent datasets $X^{(\text{train})} \in \mathbb{Z}_{\geq 0}^{n \times p}$ and $X^{(\text{test})} \in \mathbb{Z}_{\geq 0}^{n \times p}$ drawn from the same model as X, up to a parameter scaling. In this section, we review *Poisson count splitting*, which accomplishes this goal provided that $\mathbf{X}_{ij} \sim \text{Poisson}(\Lambda_{ij})$.

Algorithm 4.1 (Poisson count splitting; Neufeld et al. [2022a]) *Let $X \in \mathbb{Z}_{\geq 0}^{n \times p}$. For a chosen $M \in \mathbb{Z}^+$ and $\epsilon_1, \ldots, \epsilon_M \in (0,1)$ such that $\sum_{m=1}^{M} \epsilon_m = 1$:*

1. *Draw $\left(\mathbf{X}_{ij}^{(1)}, \mathbf{X}_{ij}^{(2)}, \ldots, \mathbf{X}_{ij}^{(M)}\right) \mid \mathbf{X}_{ij} = X_{ij} \sim \text{Multinomial}\left(X_{ij}, \epsilon_1, \ldots, \epsilon_M\right)$.*

2. *For $m = 1, \ldots, M$, let $\mathbf{X}^{\text{test,m}} = \mathbf{X}^{(m)}$ and let $\mathbf{X}^{\text{train,m}} = \mathbf{X} - \mathbf{X}^{\text{test,m}}$.*

Theorem 4.1 (Independence under Poisson count splitting) *Let $X \in \mathbb{Z}_{\geq 0}^{n \times p}$ be a dataset such that the entries X_{ij} are realizations of $\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} \text{Poisson}(\mu_{ij})$. If we apply Algorithm 4.1 to this data, the following holds for $m = 1, \ldots, M$:*

1. *$\mathbf{X}_{ij}^{\text{train,m}} \sim \text{Poisson}((1 - \epsilon_m)\mu_{ij})$,*

2. *$\mathbf{X}_{ij}^{\text{test,m}} \sim \text{Poisson}(\epsilon_m \mu_{ij})$,*

3. *$\mathbf{X}^{\text{train,m}}$ and $\mathbf{X}^{\text{test,m}}$ are independent.*

The original Poisson count splitting proposal of Neufeld et al. [2022a] assumed $M = 2$: Theorem 4.1 is a direct extension. Theorem 4.1 follows from the well-known binomial thinning property of the Poisson distribution (see Durrett 2019, Section 3.7.2).

Sarkar and Stephens [2021] and Neufeld et al. [2022a] apply Poisson count splitting to overcome the challenges arising in Examples 1 and 2 of Section 4.1, under the assumption

that the scRNA-seq data follows a Poisson distribution. Unfortunately, the Poisson assumption is necessary to achieve independence in part 3 of Theorem 4.1. If the data instead follow a negative binomial distribution, then $X^{(\text{train})}$ and $X^{(\text{test})}$ are correlated, and Poisson count splitting will fail to provide a valid approach for model evaluation or inference (see Proposition 2 in Neufeld et al. 2022a). In Section 4.3, we will recover this independence under the assumption that the elements of X_{ij} are independent negative binomial random variables.

4.2.2 Negative binomial models for scRNA-seq data

Throughout this chapter, we let $\text{NB}(\mu, b)$ denote the negative binomial distribution with mean μ and variance $\mu + \frac{\mu^2}{b}$, for $\mu > 0$ and $b > 0$. Note that $\mathbf{y} \sim \text{NB}(\mu, b)$ if $\mathbf{y} \mid \tau \sim \text{Poisson}(\mu\tau)$ and $\tau \sim \text{Gamma}(b, b)$. This is slightly different than the parameterization of the negative binomial distribution used in Neufeld et al. [2023a], but is the one commonly used for scRNA-seq data.

Because the variance $\mu + \frac{\mu^2}{b}$ is always strictly larger than the mean μ, the negative binomial model is overdispersed relative to the Poisson model. This motivates its use in the analysis of RNA sequencing data, where the data are non-negative integers with excess variance relative to the Poisson distribution [Choudhary and Satija, 2022, Hafemeister and Satija, 2019, Sarkar and Stephens, 2021]. We refer to the parameter b as the overdispersion parameter. As $b \to \infty$, the negative binomial distribution approaches the Poisson distribution, and so the negative binomial model includes the Poisson model as a special case.

In the scRNA-seq literature, it is common to assume that each gene, but not each cell, has its own overdispersion parameter [Hafemeister and Satija, 2019, Love et al., 2014]. Thus, in what follows, we will assume that $X_{ij} \sim \text{NB}(\mu_{ij}, b_j)$.

4.3 Negative binomial count splitting

4.3.1 Algorithm and main result

We now introduce negative binomial count splitting and state the key result.

Algorithm 4.2 (Negative binomial count splitting) *Let $X \in \mathbb{Z}_{\geq 0}^{n \times p}$. For a chosen $M \in \mathbb{Z}^+$, $b'_j \geq 0$ for $j = 1, \ldots, p$, and $\epsilon_1, \ldots, \epsilon_m \in (0,1)$ such that $\sum_{m=1}^{M} \epsilon_m = 1$:*

1. *Draw $\left(\mathbf{X}_{ij}^{(1)}, \mathbf{X}_{ij}^{(2)}, \ldots, \mathbf{X}_{ij}^{(M)}\right) \mid \mathbf{X}_{ij} = X_{ij} \sim \text{DirichletMultinomial}\left(X_{ij}, \epsilon_1 b'_j, \epsilon_2 b'_j, \ldots, \epsilon_m b'_j\right)$.*

2. *For $m = 1, \ldots, M$, let $\mathbf{X}^{\text{test,m}} = \mathbf{X}^{(m)}$ and let $\mathbf{X}^{\text{train,m}} = \mathbf{X} - \mathbf{X}^{(m)}$.*

The marginals of a Dirichlet-multinomial distribution are beta-binomial, and so in the $M = 2$ case Algorithm 4.2 says to draw $\mathbf{X}_{ij}^{\text{train}} \mid \mathbf{X}_{ij} = X_{ij} \sim \text{BetaBinomial}(X_{ij}, \epsilon b'_j, (1-\epsilon) b'_j)$. While binomial thinning has appeared in numerous papers as a way to construct training and test sets from count-valued data [Neufeld et al., 2022a, Sarkar and Stephens, 2021, Chen et al., 2021, Leiner et al., 2022], to our knowledge beta-binomial thinning has only been used for this purpose by Neufeld et al. [2023a]. The beta-binomial thinning operator has appeared in the time series literature at least as far back as McKenzie [1986] for constructing autoregressive processes with negative binomial marginal distributions. The following result, which appeared in the case where $M = 2$ in the context of autoregressive processes in Joe [1996], tells us what happens when $\mathbf{X}_{ij} \sim \text{NB}(\mu_{ij}, b_j)$ and we apply Algorithm 4.2 with $b'_j = b_j$.

Theorem 4.2 (Independence under negative binomial count splitting) *If $\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} \text{NB}(\mu_{ij}, b_j)$ and we apply Algorithm 4.2 with $b'_j = b_j$, then for any $m = 1, \ldots, M$:*

1. *$\mathbf{X}_{ij}^{\text{train,m}} \sim \text{NB}((1-\epsilon_m)\mu_{ij}, (1-\epsilon_m) b_j)$.*

2. *$\mathbf{X}_{ij}^{\text{test,m}} \sim \text{NB}(\epsilon_m \mu_{ij}, \epsilon_m b_j)$.*

3. *$\mathbf{X}^{\text{train,m}}$ and $\mathbf{X}^{\text{test,m}}$ are independent.*

A proof of Theorem 4.2 is included in Appendix C.2.1. The intuition behind this result is as follows. As explained in Section 4.2.2, if $\mathbf{X}_{ij} \sim \text{Poisson}(\boldsymbol{\mu}_{ij})$, and $\boldsymbol{\mu}_{ij}$ itself follows a gamma distribution with mean μ_{ij}, then $\mathbf{X}_{ij}$ marginally follows a negative binomial distribution with

mean μ_{ij}. Thus, the overdispersion in $\mathbf{X}_{ij}$ relative to a Poisson distribution can be interpreted as coming from randomness in the mean parameter. Drawing $(\mathbf{X}_{ij}^{(1)}, \ldots, \mathbf{X}_{ij}^{(M)}) \mid \mathbf{X}_{ij} = X_{ij}$ from a Dirichlet multinomial distribution, as in Algorithm 4.2, is the same as first drawing $(\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_m)$ from a Dirichlet$(\epsilon_1 b'_j, \ldots, \epsilon_m b'_j)$ distribution and then letting $(\mathbf{X}_{ij}^{(1)}, \ldots, \mathbf{X}_{ij}^{(M)}) \mid \mathbf{X}_{ij} = X_{ij} \sim$ Multinomial$(X_{ij}, \boldsymbol{\epsilon}_1 b', \ldots, \boldsymbol{\epsilon}_m b')$. Thus, to accommodate the overdispersion in $\mathbf{X}_{ij}$ relative to the Poisson distribution, we add additional randomness to the sampling process by making the parameters $(\epsilon_1, \ldots, \epsilon_m)$ from Algorithm 4.1 random variables.

The bottom line is as follows: if we believe that $\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} \text{NB}(\mu_{ij}, b_j)$ and we know the true values b_j, then a direct extension of Poisson count splitting is available.

4.3.2 The role of the parameter b'

Theorem 4.2 requires that we apply Algorithm 4.2 with the correct value of the overdispersion parameter; i.e. that $X_{ij} \sim \text{NB}(\mu_{ij}, b_j)$ and we choose $b'_j = b_j$.

In this section, we consider what happens when we use the wrong value for this parameter; i.e. $X_{ij} \sim \text{NB}(\mu_{ij}, b_j)$ with $b'_j \neq b_j$.

We first consider what happens when we use $b'_j = \infty$. Drawing $\left(\mathbf{X}_{ij}^{(1)}, \mathbf{X}_{ij}^{(2)}, \ldots, \mathbf{X}_{ij}^{(M)}\right) \mid \mathbf{X}_{ij} = X_{ij} \sim \text{DirichletMultinomial}\left(X_{ij}, \epsilon_1 b'_j, \epsilon_2 b'_j, \ldots, \epsilon_M b'_j\right)$ is the same as first drawing $(p_1, \ldots, p_M)$ from a
Dirichlet $\left(\epsilon_1 b'_j, \epsilon_2 b'_j, \ldots, \epsilon_M b'_j\right)$ distribution and then drawing $\left(\mathbf{X}_{ij}^{(1)}, \mathbf{X}_{ij}^{(2)}, \ldots, \mathbf{X}_{ij}^{(M)}\right) \mid \mathbf{X}_{ij} = X_{ij} \sim \text{Multinomial}\left(X_{ij}, p_1, p_2, \ldots, p_M\right)$. Thus, Algorithm 4.2 can be seen as a version of Algorithm 4.1 where the parameters for the multinomial sampling are random variables. When $b'_j = \infty$, each p_m is a point mass at ϵ_m. Thus, when $b'_j = \infty$, Algorithm 4.2 reduces to Algorithm 4.1. The following theorem is from Neufeld et al. [2022a].

Theorem 4.3 (Poisson count splitting of negative binomial random variables) *If* $\mathbf{X}_{ij} \sim \text{NB}\left(\mu_{ij}, b_j\right)$ *and we apply Algorithm 4.2 with* $b'_j = \infty$*, then:*

1. $\mathbf{X}_{ij}^{\text{train,m}} \sim \text{NB}((1 - \epsilon_m)\mu_{ij}, b_j)$.

2. $\mathbf{X}_{ij}^{\text{test,m}} \sim \text{NB}(\epsilon_m \mu_{ij}, b_j)$.

3. $\text{Cor}(\mathbf{X}_{ij}^{\text{train,m}}, \mathbf{X}_{ij}^{\text{test,m}}) = \frac{\sqrt{\epsilon_m(1-\epsilon_m)}}{\sqrt{\frac{b_j^2}{\mu_{ij}^2} + \frac{b_j}{\mu_{ij}} + \epsilon_m(1-\epsilon_m)}}$.

Theorem 4.3 says that while applying Poisson count splitting (or negative binomial count splitting with $b_j' = \infty$) on data from a negative binomial distribution yields training and test sets that follow the same model as the full data up to a parameter scaling, these datasets are positively correlated. The positive correlation increases as the true value of b_j decreases, and decreases to 0 as $b_j \to \infty$. Moreover, we see from Theorem 4.3 that the overdispersion parameters (and thus the variances) of $\mathbf{X}_{ij}^{\text{train,m}}$ and $\mathbf{X}_{ij}^{\text{test,m}}$ are too small relative to Theorem 4.2. Thus, by failing to put enough noise into our sampling process, applying Poisson count splitting to negative binomial data results in training and test sets that are not as noisy as they should be, leading to positive correlation between them.

We now consider the more general case of finite b_j', which is included (under a different parameterization) in Neufeld et al. [2023a].

Theorem 4.4 *If* $\mathbf{X}_{ij} \sim \text{NB}(\mu_{ij}, b_j)$ *and we apply Algorithm 4.2 with parameter* b_j'*:*

1. $E[\mathbf{X}_{ij}^{\text{train,m}}] = (1-\epsilon_m)\mu_{ij}$ *and* $E[\mathbf{X}_{ij}^{\text{test,m}}] = \epsilon_m \mu_{ij}$.

2. $\text{Var}(\mathbf{X}_{ij}^{\text{train,m}}) = (1-\epsilon_m)\,\text{Var}(\mathbf{X}_{ij}) + \epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\left(\frac{b_j+1}{b_j'+1} - 1\right)$ *and* $\text{Var}(X_{ij}^{\text{test,m}}) = \epsilon_m\,\text{Var}(\mathbf{X}_{ij}) + \epsilon(1-\epsilon)\frac{\mu_{ij}^2}{b_j}\left(\frac{b_j+1}{b_j'+1} - 1\right)$.

3. $\text{Cov}(\mathbf{X}_{ij}^{\text{train,m}}, \mathbf{X}_{ij}^{\text{test,m}}) = \epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\left(1 - \frac{b_j+1}{b_j'+1}\right)$.

Theorem 4.4 is proved in Appendix C.2.3. Unlike in Theorem 4.3, the training and test sets that result from applying Algorithm 4.2 with arbitrary values for b_j' do not necessarily follow negative binomial distributions.

The first statement of Theorem 4.4 says that, regardless of the value of b_j' used, the expected value matrices for $\mathbf{X}^{\text{train,m}}$ and $\mathbf{X}^{\text{test,m}}$ are scaled by ϵ_m and $(1-\epsilon_m)$ compared to $\text{E}[\mathbf{X}]$ in (4.1). Thus, we can obtain suitable estimates of the latent space using $\mathbf{X}^{\text{train,m}}$.

The second statement of Theorem 4.4 tells us that using the wrong value for b'_j can affect how much noise is in $\mathbf{X}^{\text{train,m}}$ and $\mathbf{X}^{\text{test,m}}$. In the ideal case where $b'_j = b_j$, then $\text{Var}(\mathbf{X}_{ij}^{\text{train,m}}) = (1 - \epsilon_m)\,\text{Var}(\mathbf{X}_{ij})$ and $\text{Var}(\mathbf{X}_{ij}^{\text{test,m}}) = \epsilon_m\,\text{Var}(\mathbf{X}_{ij})$. If $b'_j > b'$, then the training and test set variances are smaller than in this ideal case, and if $b'_j < b$ then they are too large.

The third statement says that if $b'_j > b_j$, then the training and test sets are positively correlated, whereas when $b'_j < b$ the training and test sets are negative correlated. The correlation grows with the magnitude of the discrepancy between b_j and b'_j. The result is displayed and empirically confirmed in Figure 4.3.

In order to use $X^{\text{test,f}}$ to validate a model fit to $X^{\text{train,f}}$ or to do valid inference on latent variables fit to $X^{\text{train,f}}$, we need independence between $X^{\text{train,f}}$ and $X^{\text{test,f}}$. Thus, the major takeaway from Theorem 4.4 is that, when the true b_j are unknown, it is important to estimate them well. In Section 4.4, we use the well-known R package `sctransform` [Hafemeister and Satija, 2019] to estimate each b_j.

4.3.3 The role of the parameters $\epsilon_1, \ldots, \epsilon_M$

In this section, we consider the case where we used the "correct" value of b'_j and thus the results of Theorem 4.2 hold. In this setting, it is simple to show that, for a given fold m, the parameter ϵ_m governs a tradeoff between the amount of information in the training set and in the test set. This result is summarized in the following theorem, which is proved in Appendix C.2.4.

Theorem 4.5 (Information tradeoff as we vary ϵ) *If $\mathbf{X}_{ij} \sim NB(\mu_{ij}, b_j)$, then the Fisher information contained in a single datapoint X_{ij} for the parameter μ_{ij} is $I_{\mu_{ij}}(\mathbf{X}_{ij}) = \frac{b_j}{(b_j+\mu_{ij})\mu_{ij}}$. If we apply Algorithm 4.2 with $b'_j = b_j$, then:*

1. *The Fisher information contained in a single datapoint $\mathbf{X}_{ij}^{\text{train,m}}$ for the parameter μ_{ij} is $(1 - \epsilon_m)I_{\mu_{ij}}(\mathbf{X}_{ij})$.*

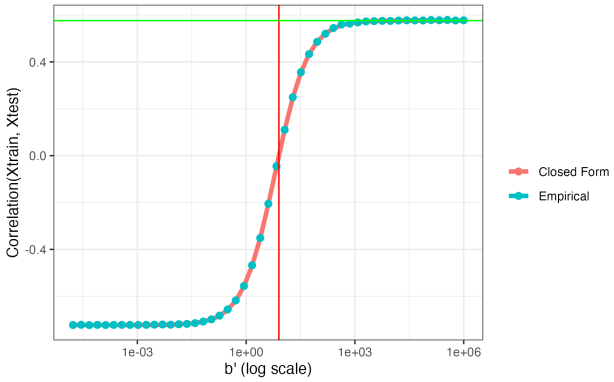


Figure 4.3: We generate $100,000$ independent realizations of $\mathbf{X}_{11} \sim \text{NB}(25, 8)$. Then, for 50 values of b'_1 ranging from 10^{-6} to 10^6, we split each of these realizations into $\mathbf{X}_{11}^{\text{train},1}$ and $\mathbf{X}_{11}^{\text{test},1}$ by applying Algorithm 4.2 with $b'_1, M = 2, \epsilon_1 = 0.3, \epsilon_2 = 0.7$. We display the sample correlation between the $100,000$ realizations of $\mathbf{X}_{11}^{\text{train},1}$ and $\mathbf{X}_{11}^{\text{test},1}$ as a function of b'_1. The pink line shows the theoretical values computed using Theorem 4.4. The horizontal green line displays $b'_1 = \infty$ given by Theorem 4.2. The vertical red line displays where $b'_1 = b_1$. As expected, both the empirical and theoretical correlations are 0 at this point.

2. *The Fisher information contained in a single datapoint* $\mathbf{X}_{ij}^{\text{test,m}}$ *for the parameter* μ_{ij} *is* $\epsilon_m I_{\mu_{ij}}(\mathbf{X}_{ij})$.

We will see in Section 4.4 that the ideal choice of $\epsilon_1, \ldots, \epsilon_M$ can depend on the application at hand.

4.4 Simulation Study

In this section, we apply negative binomial count splitting to two specific problems, which arise when Example 1 and Example 2 from Section 4.1 are instantiated in the setting where our latent variables are indicator variables for K discrete cell types and we estimate these latent variables using k-means clustering.

Problem 1: We wish to select the number of clusters that best fit the observed data.

Problem 2: We wish to estimate two cell types and then test each gene for differential expression across the two estimated clusters.

While highly specialized methods exist for each of these tasks, our goal is to show that negative binomial count splitting is one flexible framework that can be easily used to answer both of these questions.

After introducing our simulation setup in Section 4.4.1, we address Problem 1 in Section 4.4.2 and address Problem 2 in Section 4.4.3.

4.4.1 Data generating mechanism

We generate datasets with n cells, p genes, and K true cell types. Each gene has a baseline expression level $\exp(\beta_{j0})$ where $\beta_{j0} \overset{\text{i.i.d.}}{\sim} N(0,1)$ for $j = 1, \ldots, p$.

For each dataset, we assign each cell to one of the K clusters with equal probability. The first column of the latent variable matrix $L \in \mathbb{R}^{n \times K}$ stores ones, and the rest of columns are indicators for clusters $2, \ldots, K$. The matrix $\beta \in \mathbb{R}^{p \times K}$ stores $\beta_{10}, \ldots, \beta_{p0}$ in the first column. When $K > 1$, $\beta_{j2} = \beta^*$ for the first 5% of the genes, and the rest of the entries in the second column of β are 0. If $K > 2$, then $\beta_{j3} = \beta^*$ for the next 5% of the genes, and all other entries are 0. We continue filling in the β matrix in this manner until all K clusters have been accounted for. We consider different values of β^* for different datasets, but within a dataset we always use the same value of β^* such that all K clusters are equally easy to detect. Finally, we let $\log(\Lambda) = L\beta^\top$.

To generate overdispersion parameters for each of the j genes, we let $\bar{\Lambda}_j = \frac{1}{n}\sum_{i=1}^{n} \Lambda_{ij}$ be the average expression for the jth gene, and we set $b_j = \frac{\bar{\Lambda}_j}{\text{overdisp}}$. This means that $Var(X_{ij}) = \Lambda_{ij}\left(1 + \frac{\Lambda_{ij}}{b_j}\right) \approx \Lambda_{ij}\,(1 + \text{overdisp})$. We consider datasets where overdisp $= 1$ and settings where overdisp $= 5$; we call these settings "mild overdispersion" and "severe overdispersion". The assumption that the parameter b_j is a function of the average expression is motivated by Choudhary and Satija [2022], Hafemeister and Satija [2019] and Love et al. [2014].

Finally, we let $\mathbf{X}_{ij} \sim \text{NB}(\Lambda_{ij}, b_j)$. Note that, compared to (4.1), we have omitted size factors from this simulation study, as they are not the focus of this chapter.

4.4.2 Selecting the number of clusters

4.4.2.1 Methods

We now introduce the general algorithm used in this section.

Algorithm 4.3 (Estimating the number of clusters)

1. *Start with datasets* $X^{(\text{train})} \in \mathbb{Z}_{\geq 0}^{n \times p}$ *and* $X^{(\text{test})} \in \mathbb{Z}_{\geq 0}^{n \times p}$ *and parameter* $\epsilon \in (0, 1)$.

2. *For* $k = 1, \ldots, 10$*:*

   (a) *Run k-means clustering to estimate* k *clusters on* $\log(X^{(\text{train})} + 1)$. *This yields a cluster assignment* $\hat{c}_i \in \{1, \ldots, k\}$ *for* $i = 1, \ldots, n$ *in* $X^{(\text{train})}$.

   (b) *For* $i = 1, \ldots, n$ *and* $j = 1, \ldots, p$, *estimate* $E[X_{ij}^{\text{train}}]$ *with the sample mean of all the training set points assigned to the same cluster:*

$$\hat{\mu}_{ij}^{\text{train}} = \frac{1}{\sum_{i=1}^{n} \mathbf{1}\{\hat{c}_{i'} = \hat{c}_i\}} \sum_{i'=1}^{n} X_{i'j}^{\text{train}} \mathbf{1}\{\hat{c}_{i'} = \hat{c}_i\}.$$

   (c) *Using the known relationship between* $E[X_{ij}^{\text{train}}]$ *and* $E[X_{ij}^{\text{test}}]$, *estimate* $E[X_{ij}^{\text{test}}]$ *as:*

$$\hat{\mu}_{ij}^{\text{test}} = \frac{1 - \epsilon}{\epsilon} \hat{\mu}_{ij}^{\text{train}}.$$

   (d) *Compute the within-cluster mean squared error on the test set as:*

$$MSE(k) = \frac{1}{n \times p} \sum_{i=1}^{n} \sum_{j=1}^{p} \left(\log\left(X_{ij}^{\text{test}} + 1\right) - \log\left(\hat{\mu}_{ij}^{\text{test}} + 1\right) \right)^2. \tag{4.2}$$

We consider the following variations on Algorithm 4.3.

Naive method: Run Algorithm 4.3 with $X^{(\text{train})} = X^{(\text{test})} = X$ and $\epsilon = 0.5$.

Poisson count splitting (PCS): Obtain $X^{(\text{train})} = X^{\text{train},1}$ and $X^{(\text{test})} = X^{\text{test},1}$ by running Algorithm 4.1 on the data X with $M = 2$ and parameters $(\epsilon, 1-\epsilon)$. Then run Algorithm 4.3 using $X^{(\text{train})}, X^{(\text{test})}$, and ϵ.

Negative binomial count splitting, known b (NBCS-known): Obtain $X^{(\text{train})} = X^{\text{train},1}$ and $X^{(\text{test})} = X^{\text{test},1}$ by running Algorithm 4.2 on matrix X with $M = 2$, $(\epsilon, 1-\epsilon)$, and $(b'_1, \ldots, b'_p) = (b_1, \ldots, b_p)$. Then run Algorithm 4.3 using $X^{(\text{train})}, X^{(\text{test})}$, and ϵ.

Negative binomial count splitting, estimated b (NBCS-estimated): First use the R package `sctransform` [Hafemeister and Satija, 2019] to estimate $b_1, \ldots, b_p$. Details are given in Appendix C.3. Then obtain $X^{(\text{train})} = X^{\text{train},1}$ and $X^{(\text{test})} = X^{\text{test},1}$ by running Algorithm 4.2 on matrix X with $M = 2$, $(\epsilon, 1-\epsilon)$, and $(b'_1, \ldots, b'_p) = (\hat{b}_1, \ldots, \hat{b}_p)$. Then proceed as in NBCS-known.

We note that PCS could also be called NBCS with $b' = \infty$. We now extend both versions of NBCS to perform cross validation.

Negative binomial cross validation, known b (NBCV-known): Obtain $(X^{(1)}, \ldots, X^{(M)})$ by running Algorithm 4.2 on matrix X with $M = 10$, $\epsilon_m = \frac{1}{M}$ for $m = 1, \ldots, M$, and $(b'_1, \ldots, b'_p) = (b_1, \ldots, b_p)$. For $m \in 1, \ldots, M$, apply Algorithm 4.3 with $X^{(\text{train})} = X^{\text{train,f}}$, $X^{(\text{test})} = X^{\text{test,f}}$, and $\epsilon = \frac{M-1}{M}$. For each value of k, save the total MSE summed across the M folds.

Negative binomial cross validation, estimated b (NBCV-estimated): As above, but first estimate $b_1, \ldots, b_p$ by using the `R` package `sctransform` to the full data X, and use these estimated values when applying Algorithm 4.2.

We already showed in Section 4.1 that the naive method fails to provide a viable solution to Problems 1 and 2, but we include it here for the sake of comparison. We do not consider sample splitting in this section, as we already saw in Section 4.1 that it fails to provide a viable solution to Problems 1 and 2, and it cannot be described as naturally using Algorithm 4.3.

4.4.2.2 Results

We first generate 1,000 datasets with $n = 1000$ and $p = 1000$ and $\beta^* = 1.5$ for each $K \in \{1, 3, 5\}$ and each overdispersion setting described in Section 4.4.1. For each dataset, we consider the naive method, Poisson count splitting, NBCS-known, and NBCS-estimated. For each count splitting method, we fix $\epsilon = 0.5$. We plot the average MSE over the 1000 datasets, defined in (4.2), as a function of k. To facilitate comparisons between methods, the y-axis of Figure 4.4 has been scaled such that the MSE for each method ranges from 0 to 1, and is displayed on a log scale.

Figure 4.4 shows that, regardless of the true value of K or the amount of overdispersion, we see that the MSE for the naive method decreases monotonically with k. Thus, we cannot simply select the value of k that minimizes the loss function, and must instead search for a bend or an "elbow" in the MSE plot. This heuristic method fails to provide a clear answer for the number of clusters we should select in the examples above. We see that PCS performs well (the loss function is minimized when $K = k$) under mild overdispersion, but under severe overdispersion its performance approaches that of the naive method. This is as expected from Theorem 4.3, where we saw that the correlation between $X^{(\text{train})}$ and $X^{(\text{test})}$ under PCS increases as the amount of overdispersion increases. Finally, we see that both versions of NBCS have loss functions that are minimized when $K = k$, regardless of the true value of K or the amount of overdispersion in the data.

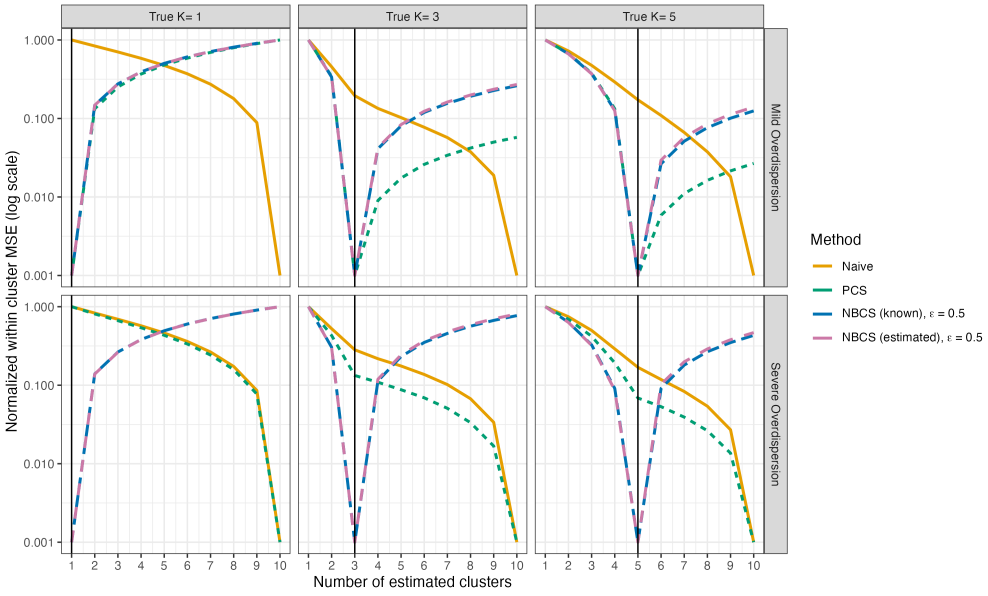


Figure 4.4: The average within cluster MSE over 1000 datasets for four of the methods described in Section 4.4.2.1, scaled to have range between 0 and 1.

Figure 4.4 demonstrates that NBCS provides a useful solution to Problem 1. Since the loss function will be minimized at $K = k$ when the signal is sufficiently strong, we can select the number of clusters by selecting the value of k that minimizes the loss function. As the naive method and PCS do not provide a useful solution to Problem 1, we do not consider them for the remainder of this section.

We next explore the role of the parameter ϵ in NBCS. We generate 2,000 datasets with mild overdispersion with $n = 500$ and $p = 40$ for $K = 5$ and for β^* values ranging from 2 to 6. For values of ϵ ranging from 0 to 1, we perform NBCS-known. For each dataset and each value of ϵ, we consider three metrics.

We first consider the adjusted Rand Index [Hubert and Arabie, 1985] between the true clusters and those estimated using $X^{(\text{train})}$ when $k = K$. The left panel of Figure 4.5 displays the average adjusted Rand Index as a function of ϵ. For a given signal strength, the average adjusted Rand Index increases with ϵ because a large value of ϵ means that we use more of the information in our data in the cluster estimation phase of Algorithm 4.3 (see Theorem 4.5).

We then consider whether or not the MSE is minimized at $K = k$. We first consider this metric only among datasets where the adjusted Rand Index between the true clusters and the estimated clusters when $K = k$ exceeds 0.8; i.e. datasets where we were able to estimate "good" clusters. This metric is shown in the center panel of Figure 4.5. We can see that this metric decreases with ϵ (center panel of Figure 4.5). Given that we estimated good clusters, a large value of ϵ simply means that there is less information in the test set for us to evaluate the clusters. The right panel of Figure 4.5 shows the overall proportion of datasets for which we select $K = k$. We can see that the optimal value of ϵ depends on the true signal strength, but that it involves a tradeoff between choosing ϵ large enough to estimate good clusters on $X^{(\text{train})}$, but not so large that we do not leave enough information over to validate these clusters.

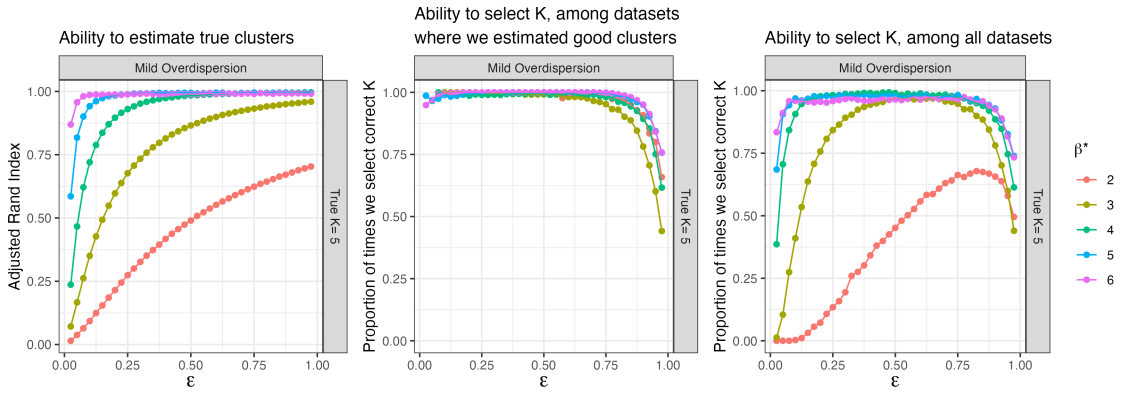


Figure 4.5: We generate 2,000 datasets with mild overdispersion, $n = 500$, $p = 40$, $K = 5$ for each of 5 values of β^* ranging from 2 to 6. For values of ϵ ranging from 0 to 1, we perform NBCS-known. *Left:* The average adjusted Rand Index between the true clusters and those estimated with $X^{(\text{train})}$ when $k = K$, as a function of ϵ. *Center:* The proportion of datasets for which the MSE is minimized at $K = k$, only considering datasets for which the adjusted Rand Index between the true clusters and the estimated clusters when $K = k$ exceeds 0.8. *Right:* The overall proportion of datasets for which the MSE is minimized at $K = k$.

Finally, we look at the role of multiple folds by comparing NBCS-known with $\epsilon = 0.9$ to NBCV-known with 10 folds. We generate 2,000 datasets where $n = 500$ and $p = 40$

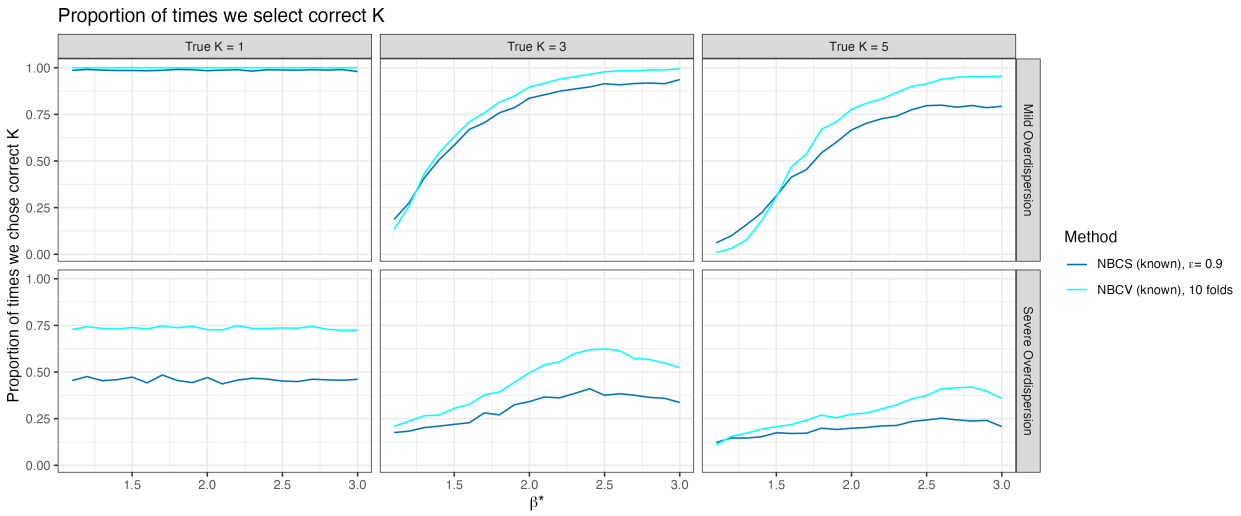


Figure 4.6: We fix $n = 500$ and $p = 40$ and we generate 200 datasets for each combination of $K \in \{1, 3, 5\}, overdisp \in \{1, 5\}$, and β^* values ranging from 0.1 to 3. For each value of β^*, we show the proportion of datasets for which our method identified the correct value of K.

for values of β^* ranging from 1 to 6. As both methods use 90% of the information in the data for training and 10% for testing, these methods will have the same average MSE curves when the MSE is averaged over many datasets. However, for a given dataset, we expect NBCV-known to outperform NBCS-known, because by averaging the MSE over 10 folds of data the former obtains a lower-variance estimate of model quality, which allows us to more consistently select the correct estimate for K (Figure 4.6).

4.4.3 Testing for differential expression

4.4.3.1 Methods

For simplicity, in this section we let $K = 2$ and we always estimate two clusters on the data. Thus, our focus is no longer on estimating the number of clusters, but rather on analyzing the clusters given that we have estimated them.

In this section, we use the following algorithm, which has several variations.

Algorithm 4.4 (Testing for differential expression.)

1. *Start with datasets* $X^{(\text{train})} \in \mathbb{Z}_{\geq 0}^{n \times p}$ *and* $X^{(\text{test})} \in \mathbb{Z}_{\geq 0}^{n \times p}$.

2. *Apply k-means clustering with* $k = 2$ *to estimate clusters on* $\log(X^{(\text{train})} + 1)$. *This yields a cluster assignment* $\hat{c}_i \in \{0, 1\}$ *for* $i = 1, \ldots, n$.

3. *For* $j = 1, \ldots, p$, *fit a negative binomial GLM of* X_j^{test} *on* $\hat{c}$. *Report the Wald p-value for the slope coefficient.*

We consider the following variations on Algorithm 4.4.

Naive method: Run Algorithm 4.4 with $X^{(\text{train})} = X^{(\text{test})} = X$.

Poisson count splitting (PCS): Obtain $X^{(\text{train})} = X^{\text{train},1}$ and $X^{(\text{test})} = X^{\text{test},1}$ by running Algorithm 4.1 on the data X with $M = 2$ and parameter $(\epsilon, 1 - \epsilon)$. Then proceed with Algorithm 4.4.

Negative binomial count splitting, known b **(NBCS-known):** Obtain $X^{(\text{train})} = X^{\text{train},1}$ and $X^{(\text{test})} = X^{\text{test},1}$ by running Algorithm 4.2 on matrix X with $M = 2$, $(\epsilon, 1 - \epsilon)$, and $(b'_1, \ldots, b'_p) = (b_1, \ldots, b_p)$. Then proceed with Algorithm 4.4.

Negative binomial count splitting, estimated b **(NBCS-estimated:)** Use the `R` package `sctransform` [Hafemeister and Satija, 2019] to estimate $b_1, \ldots, b_p$ using the full dataset X. Then proceed as in NBCS-known.

Unlike in Section 4.4.2, we do not attempt to use NBCV. We do not do splitting with $M > 2$ and we do not aggregate results across folds by interchanging the roles of the train and test sets. We leave the possibility of aggregating differential expression test statistics across multiple folds to future work. Once again, we do not consider sample splitting because we already saw in Figure 4.2 that it fails to provide a viable solution to Problem 2.

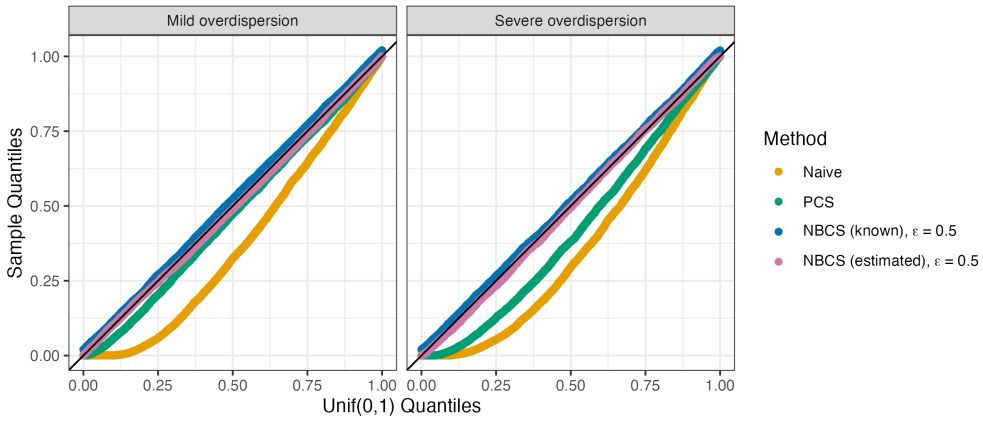


Figure 4.7: We generate $1,000$ datasets with $K = 2, n = 500$, and $p = 40$ for each overdispersion setting and for each of 16 values of β^* ranging from 0 to 3. For each dataset, we carry out the four variations of Algorithm 4.4 given in Section 4.4.3.

4.4.3.2 Results

We generate datasets using the mechanism described in Section 4.4.1 with $K = 2$, $n = 500$, and $p = 40$. Under this mechanism, the first two genes are differentially expressed across the two true clusters, but the remaining 38 genes have the same expected value across all cells (and thus are not differentially expressed). We refer to these non-differentially expressed genes as the null genes. Figure 4.7 shows uniform QQ plots of the p-values obtained from the four variations of Algorithm 4.4 for the null genes, aggregated across 1000 datasets for each of 16 β^* values. We see from Figure 4.7 that both the naive method and Poisson count splitting both fail to control the Type 1 error rate, with Poisson count splitting performing worse when overdispersion is severe. On the other hand, both versions of NBCS control the Type 1 error rate.

Finally, we explore the role of ϵ in this setting. We generate 1000 datasets where $n = 1000$

and $p = 1000$ for each of 16 values of β^* ranging from 0 to 3. For each dataset, we consider the adjusted Rand Index between the true clusters and those estimated on the training set. As in Section 4.4.2, we expect that larger values of ϵ will lead to higher adjusted Rand Indices, on average, for a given signal strength β^*. This is confirmed in the left panel of Figure 4.8.

On the other hand, given the clusters that we estimated, we expect that smaller values of ϵ will leave us with more power to detect differential expression. We define $\hat{\beta}^*$ to be the estimated GLM coefficient that we get for a gene X_j if we regress its mean vector Λ_j onto the *estimated* clusters (estimated on the training set). We note that $\hat{\beta}^* = \beta^*$ only if the estimated clusters are exactly equal to the true clusters. The right panel of Figure 4.8 plots the proportion of times that the differential expression p-value for a non-null gene was less than 0.05, as a function of $\hat{\beta}^*$ (computed for the appropriate gene and the appropriate estimated clusters). We see that, for a given value of $\hat{\beta}^*$, the power is highest when ϵ is small. This makes sense; when ϵ is small, the test set contains more information, and thus has more power to reject the null.

4.5 Application to fetal cell atlas data

In this section, we show through an analysis of real data that negative binomial count splitting is a useful solution to Example 4.3 from Section 4.1.

Cao et al. [2020] sequenced more than 4 million cells from 121 human fetal samples to create a fetal cell atlas, with a goal of organizing the cells from each of 15 different organs into cell types and cell subtypes. After preprocessing, they applied clustering to the expression data from the kidney cells to characterize the kidney cells into 9 main cell types. They then subsetted the data to include only the cluster thought to correspond to metanephric kidney cells, and applied an additional layer of clustering to divide the metanephric kidney cells into 10 cell subtypes. After this clustering process, the authors were faced with the task of validating these clusterings to convince the reader that the 9 kidney cell types and the 10 metanephric cell subtypes are reproducible; see Figure 2 of Cao et al. [2020].

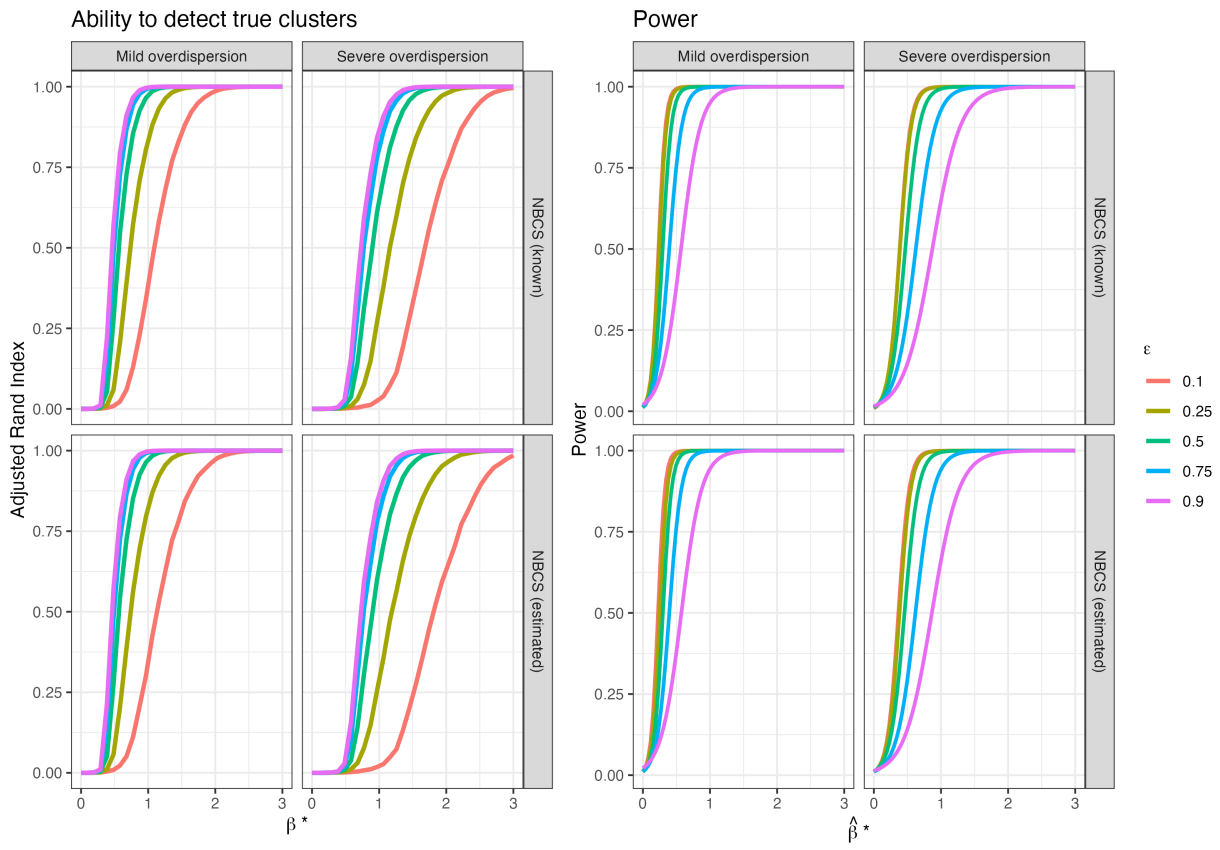


Figure 4.8: *Left:* The average adjusted Rand index between the true clusters and those estimated on the training set, plotted as a function of β^* for both overdispersion settings and for each value of ϵ. *Right:* The proportion of times that the differential expression p-value for a non-null gene was less than 0.05, as a function of the association between the gene's expected expression and the estimated clusters.

As there is no ground truth available for the main cell types or the cell subtypes, this is a difficult validation task. Cao et al. [2020] use a procedure that they call *intradataset cross-validation*, which we describe in Algorithm 4.5.

Algorithm 4.5 (Intradataset cross-validation, Cao et al. [2020])
Input: a cell-by-gene expression matrix $X \in \mathbb{Z}_{\geq 0}^{n \times p}$.

1. *After the appropriate preprocessing steps, run a clustering algorithm on the full dataset* X *to come up with estimated cell types* $\hat{L}(X)^{\text{cluster}}$.

2. *Divide the* n *cells into* 5 *folds and, for each fold* $m = 1, \ldots, 5$*:*

   (a) *Let all cells in fold* m *be the test set, and let the remaining cells be the training set.*

   (b) *Train a classifier to predict* $\hat{L}(X)_i^{\text{cluster}}$ *using* X_i *using all cells* i *in the training set.*

   (c) *Using this trained classifier, obtain predictions* $\hat{L}_{i'}^{\text{classifier}}$ *for each cell* i' *in the test set, on the basis of* $X_{i'}$.

3. *Make a confusion matrix comparing* $\hat{L}(X)^{\text{cluster}}$ *and* $\hat{L}^{\text{classifier}}$. *A diagonal confusion matrix is treated as evidence that the clusters are stable, since the cluster assignment for a given cell can be reliably recovered by the classifier, even when that particular cell was not used to train the classifier.*

In Algorithm 4.5, we additionally compute the adjusted Rand Index (ARI; [Hubert and Arabie, 1985]) as a numerical summary of the degree of agreement between $\hat{L}(X)^{\text{cluster}}$ and $\hat{L}^{\text{classifier}}$. In Section 4.1 (Figure 4.2(e)), we showed using a toy dataset that there is an issue with this procedure. Because the entire dataset X is used in Step 1 to estimate the clusters $\hat{L}(X)^{\text{cluster}}$, the test set in Step 2(c) has not truly been held out of the training process. Thus, despite the fact that the clusters estimated on the toy dataset are driven by random noise and

would not be reproducible on a new dataset, the confusion matrix output by Algorithm 4.5 is close to diagonal (see Figure 4.2(e)).

Luckily, negative binomial count splitting provides a simple alternative to Algorithm 4.5 that provides a more sensible confusion matrix in the context of the toy example (see Figure 4.2(f)). We outline the procedure in Algorithm 4.6.

Algorithm 4.6 (Intradataset cross-validation via count splitting)
Input: a cell-by-gene expression matrix $X \in \mathbb{Z}_{\geq 0}^{n \times p}$.

1. *Obtain estimates* $\hat{b}_j$ *of the gene-specific overdispersion* b_j *for* $j = 1, \ldots, p$.

2. *Apply step 1 of Algorithm 4.2 (negative binomial count splitting) with* $M = 2$, $\epsilon_1 = \epsilon_2 = 0.5$, *and* $(b'_1, \ldots, b'_p) = \hat{b}_1, \ldots, \hat{b}_p)$ *to create two folds of data* $X^{(1)}$ *and* $X^{(2)}$.

3. *After the appropriate preprocessing steps, apply a clustering algorithm to* $X^{(1)}$ *to to obtain an estimated cluster* $\hat{L}(X^{(1)})_i$ *for each cell.*

4. *After the appropriate preprocessing steps, apply a clustering algorithm to* $X^{(2)}$ *to to obtain an estimated cluster* $\hat{L}(X^{(2)})_i$ *for each cell.*

5. *Make a confusion matrix comparing* $\hat{L}(X^{(1)})$ *and* $\hat{L}(X^{(2)})$. *A diagonal confusion matrix (up to a permutation of the columns) is treated as evidence of stability, since the cells can be reliably re-assigned to the same cluster, even when independent realizations of data are used.*

In Algorithm 4.6, we additionally compute the adjusted Rand Index (ARI; [Hubert and Arabie, 1985]) as a numerical summary of the degree of agreement between $\hat{L}(X^{(1)})$ and $\hat{L}(X^{(2)})$. This is particularly useful here, as the adjusted Rand Index can handle cases where different numbers of clusters were estimated on the two datasets, such that multiple clusters from $X^{(2)}$ may correspond to a single cluster in $X^{(1)}$.

We now compare Algorithm 4.5 and Algorithm 4.6 on real data from the human fetal cell atlas of Cao et al. [2020]. We consider two versions of Algorithm 4.6. We apply a version where, in Step 1, we let each $\hat{b}_j = \infty$, which corresponds to assuming that the data are Poisson. We also apply a version where, in Step 1, we estimate each $\hat{b}_j$ using the full dataset using `sctransform`; details are given in Appendix C.4. In both cases, we use $\epsilon_1 = \epsilon_2 = 0.5$ such that $X^{(1)}$ and $X^{(2)}$ are identically distributed. Our goal is not to discover the optimal clusters in this dataset, but rather to compare Algorithm 4.5 and Algorithm 4.6 as strategies for validating clusters. As such, we do not attempt to reproduce the exact analysis from Cao et al. [2020]; we use a simplified implementation, which is described in Appendix C.4.

To start, we apply Algorithm 4.5 and both versions of Algorithm 4.6 to all $178,603$ kidney cells from the human fetal cell atlas. The results are shown in panels (a), (b), and (c) of Figure 4.9. Regardless of the algorithm used, we see diagonal confusion matrices and high adjusted rand indices, suggesting a high degree of stability of the clusters. For panels (b) and (c), we permute the columns of the matrices to make them appear as diagonal as possible, but we note that the adjusted Rand Index is invariant to these permutations.

We next apply Algorithm 4.5 and both versions of Algorithm 4.6 to the $90,876$ kidney cells that Cao et al. [2020] annotated as metanephric cells. The results are shown in panels (d), (e), and (f) of Figure 4.9. Overall, panels (e) and (f) show less stability (i.e. lower adjusted Rand Indices) than panels (b) and (c), suggesting that these supposed cell subtypes are somewhat less reproducible than the main cell types. We note that the difference between the main cell type analysis and the cell subtype analysis is least stark for Algorithm 4.5, where the double use of data causes the cell subtypes to appear more reproducible than they are. The difference between the main cell type analysis and the cell subtype analysis is most stark for the negative binomial version of Algorithm 4.6, suggesting that the Poisson version is affected by dependence between $X^{(1)}$ and $X^{(2)}$.

As the confusion matrices in Figure 4.9 visualize just a single random split of the data, we also repeat each version of Algorithm 4.6 ten times for each dataset, such that we use ten different random splits of the data. The adjusted Rand Indices resulting from the then

different splits are shown in Figure 4.9(d) and 4.9(h). We see that the adjusted Rand Indices from the cell subtype analysis are consistently lower than those from the main cell type analysis.

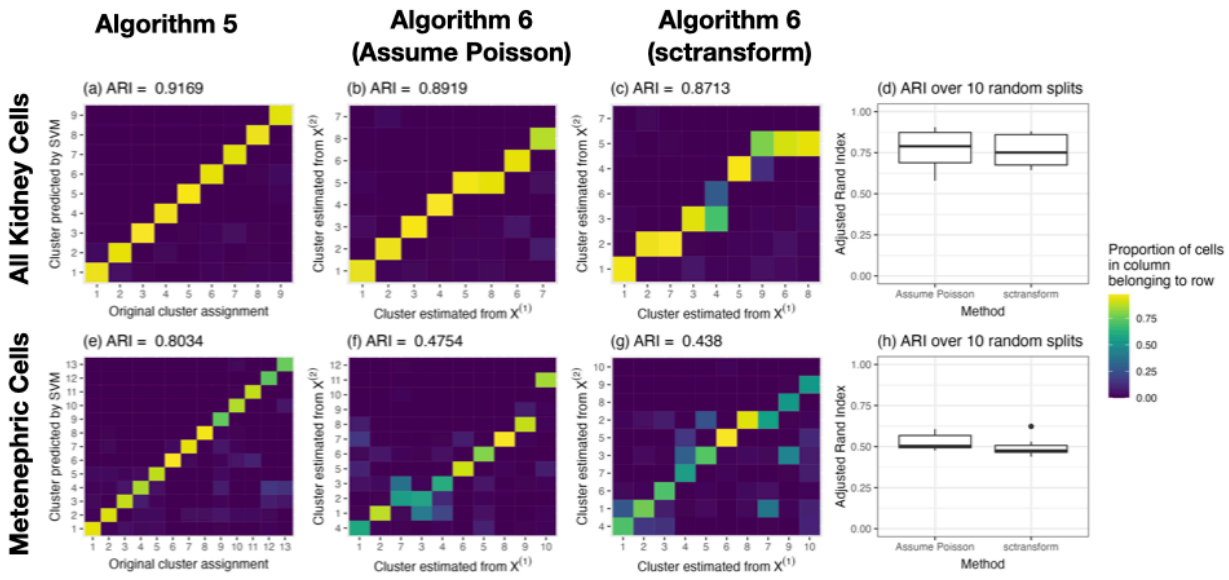


Figure 4.9: The results of applying (a) Algorithm 4.5, (b) Algorithm 4.6 assuming the data is Poisson, and (c) Algorithm 4.6 with overdispersion parameters estimated via `sctransform` to the full kidney dataset. Panels (d), (e), and (f) show the same results, but applied to the metanephric cells only.

In conclusion, Algorithm 4.6 provides a more statistically principled way to assess the stability of clusters than 4.5, and allows us to see that the cell subtypes are less stable than the main cell types.

4.6 Discussion

In this chapter, we introduced an algorithm to split negative binomial data into independent training and test sets that have the same dimensions as the original data. We showed that this algorithm has useful applications in the analysis of scRNA-seq data. This algorithm has the potential to be useful well beyond scRNA-seq data, as data are modeled as negative

binomial in a wide variety of fields. Furthermore, while we were particularly motivated by unsupervised settings in which sample splitting is not an option, negative binomial count splitting can also be used in supervised settings such as inference after variable selection in negative binomial regression. Further uses for negative binomial count splitting and other forms of data thinning are discussed in Neufeld et al. [2023a] and Dharamshi et al. [2023].

An implementation of the techniques in this chapter is available in the `R` package `countsplit`, and tutorials showing how this can be integrated with existing packages for the analysis of scRNA-seq data are available at `anna-neufeld.github.io/countsplit.tutorials`.

Chapter 5

DATA THINNING FOR CONVOLUTION-CLOSED DISTRIBUTIONS

The contents of this chapter are contained in [Neufeld et al., 2023a], which is available as a preprint on arXiv.

5.1 Introduction

As scientists fit increasingly complex models to their data, there is an ever-growing need for out-of-box methods that can be used to validate these models. In many settings, the most natural option is sample splitting, in which the observations are split into a training set, used to fit a model, and a test set, used to validate it [Hastie et al., 2009]. Sample splitting can also be applied to conduct inference after model selection [**?**]. However, in some settings, sample splitting is neither applicable nor desirable, and alternative approaches are necessary. In this chapter, we consider an alternative approach that splits a single observation X into independent parts that follow the same distribution as X.

It has recently been shown that we can split $X \sim N(\mu, \sigma^2)$ into two independent Gaussian random variables [Rasines and Young, 2022, Leiner et al., 2022, Oliveira et al., 2021], and $X \sim \text{Poisson}(\lambda)$ into independent Poisson random variables [Neufeld et al., 2022a, Leiner et al., 2022]. However, outside of these two distributions, no proposals are available to split a random variable into independent parts that follow the same distribution as the original random variable. Leiner et al. [2022] proposed a general-purpose approach to decompose X into two parts, $X^{(1)}$ and $X^{(2)}$, such that (i) $X^{(1)}$ and $X^{(2)}$ can together be used to reconstruct X, and (ii) the joint distribution of $\left(X^{(1)}, X^{(2)}\right)$ is tractable. However, the resulting $X^{(1)}$ and $X^{(2)}$ are not independent, and typically do not follow the same distribution as X. These

considerations complicate the application of data fission to model validation. We elaborate on these points in Section D.1 of the supplementary materials.

In this chapter, we propose data thinning, a recipe for decomposing an observation X into two parts, $X^{(1)}$ and $X^{(2)}$, such that (i) $X = X^{(1)} + X^{(2)}$, (ii) $X^{(1)}$ and $X^{(2)}$ are independent, and (iii) $X^{(1)}$ and $X^{(2)}$ follow the same distribution as X, up to a (known) scaling of a parameter. Critically, properties (ii) and (iii) guarantee that this decomposition is useful in applied settings. For instance, to evaluate the suitability of a model for X, we can fit it to $X^{(1)}$ (which follows the same distribution as X), and can validate it using $X^{(2)}$ (which also follows the same distribution, and furthermore is independent of $X^{(1)}$). Our recipe can be applied to any distribution that is convolution-closed [Joe, 1996]: this includes the multivariate Gaussian, Poisson, negative binomial, Gamma, binomial, and multinomial distributions, among others. Thus, our work drastically expands the set of distributions that can be split into independent parts, and provides a unified lens through which to view seemingly unrelated approaches. Furthermore, data thinning can be used to decompose X into more than two independent random variables.

We illustrate our proposal with the following example, which shows that a Gamma random variable can be thinned into M independent Gamma random variables.

Example 5.1 (Gamma decomposition into M components, data thinning) *Suppose that* $X \sim \text{Gamma}(\alpha, \beta)$*, where* β *is unknown. We take* $(X^{(1)}, \ldots, X^{(M)}) = XZ$*, where* $Z \sim \text{Dirichlet}(\alpha/M, \ldots, \alpha/M)$*. Then* $X^{(1)}, \ldots, X^{(M)}$ *are mutually independent, they sum to* X*, and each is marginally drawn from a* $\text{Gamma}(\alpha/M, \beta)$ *distribution.*

In other words, data thinning allows us to decompose a $\text{Gamma}(\alpha, \beta)$ random variable, for which β is unknown, into M independent Gamma random variables, $X^{(1)}, \ldots, X^{(M)}$. Therefore, fitting a model to $X - X^{(m)}$ and validating it using $X^{(m)}$ is straightforward.

Data thinning can be applied to any problem for which sample splitting might be considered. In this chapter, we specifically focus on its use in model assessment and validation. While this approach is equally applicable to both supervised and unsupervised learning, in

our numerical studies we focus on the unsupervised learning setting, in which the usual cross-validation via sample splitting approach cannot be directly applied [see, e.g. Owen and Perry, 2009, Fu and Perry, 2020]. We show that cross-validation via data thinning provides an attractive alternative.

5.2 The data thinning proposal

5.2.1 A review of convolution-closed distributions

We begin by defining a convolution-closed distribution [Joe, 1996, Jørgensen and Song, 1998].

Definition 5.1 (Convolution-closed) *Let F_λ denote a distribution indexed by a parameter λ in parameter space Λ. Let $X' \sim F_{\lambda_1}$ and $X'' \sim F_{\lambda_2}$ with $X' \perp\!\!\!\perp X''$. If $X' + X'' \sim F_{\lambda_1+\lambda_2}$ whenever $\lambda_1 + \lambda_2 \in \Lambda$, then F_λ is* convolution-closed *in the parameter λ.*

Many well-known distributions are convolution-closed. While the Poisson(λ) distribution is convolution-closed in its single parameter λ and the $\mathrm{N}(\mu, \sigma^2)$ distribution is convolution-closed in the two-dimensional parameter (μ, σ^2), other distributions, such as the Gamma, are convolution-closed in just one parameter with the other parameter(s) held fixed. Table 5.1 provides details about some well-known convolution-closed distributions.

Remark 5.1 (Expectation is linear in λ) *Let F_λ for $\lambda \in \Lambda$ be a convolution-closed distribution. By definition, if $X' \sim F_{\lambda_1}$ and $X'' \sim F_{\lambda_2}$ and $\lambda_1 + \lambda_2 \in \Lambda$, then $X' + X'' \sim F_{\lambda_1+\lambda_2}$. If these distributions have first moments, then $E[X' + X''] = E[X'] + E[X'']$. Thus, outside of contrived counterexamples, $E[X]$ for $X \sim F_\lambda$ is a linear function of the parameter λ. We say distributions whose expectation is linear in λ (e.g. all distributions in Table 5.1) satisfy the* linear expectation property*.*

For a convolution-closed distribution F_λ, suppose that $X' \sim F_{\lambda_1}$ and $X'' \sim F_{\lambda_2}$ with $X' \perp\!\!\!\perp X''$. Let $G_{\lambda_1,\lambda_2,x}$ denote the conditional distribution of $X' \mid X' + X'' = x$. The density of the distribution $G_{\lambda_1,\lambda_2,x}$ can be written down for any F_λ with a known density

Table 5.1: A partial list of convolution-closed distributions. The last two rows contain multivariate distributions. The results in each row are easily verifiable. The generalized Poisson and Tweedie distributions are written in their additive exponential dispersion family parameterization; see Jørgensen and Song [1998] for details

Distribution	Notes
$X \sim \text{Poisson}(\lambda)$, where $\text{E}[X] = \lambda$ and $\text{Var}(X) = \lambda$.	Convolution-closed in λ.
$X \sim \text{N}(\mu, \sigma^2)$, where $\text{E}[X] = \mu$ and $\text{Var}[X] = \sigma^2$.	Convolution-closed in (μ, σ^2).
$X \sim \text{NegativeBinomial}(r, p)$, where $\text{E}[X] = r\frac{1-p}{p}$ and $\text{Var}[X] = r\frac{1-p}{p^2}$.	Convolution-closed in r if p is fixed.
$X \sim \text{Gamma}(\alpha, \beta)$, where $\text{E}[X] = \frac{\alpha}{\beta}$ and $\text{Var}(X) = \frac{\alpha}{\beta^2}$.	Convolution-closed in α if β is fixed.
$X \sim \text{Binomial}(r, p)$, where $\text{E}[X] = rp$ and $\text{Var}(X) = rp(1-p)$.	Convolution-closed in r if p is fixed.
$X \sim \text{InverseGaussian}(\mu w, \lambda w^2)$ with $\text{E}[X] = \mu w$ and $\text{Var}(X) = \frac{w^3\mu^3}{w^2\lambda} = \frac{w\mu^3}{\lambda}$.	Convolution-closed in w if μ and λ are fixed.
$X \sim \text{GeneralizedPoisson}(\lambda, \theta)$, see Jørgensen and Song [1998] for parameterization.	Convolution-closed in λ if θ is fixed.
$X \sim \text{Tweedie}_p(\lambda, \theta)$, see Jørgensen and Song [1998] for parameterization.	Convolution-closed in λ if θ and p are fixed.
$X \sim \text{N}_k(\mu, \Sigma)$, with $\text{E}[X] = \mu$ and $\text{Var}(X) = \Sigma$.	Convolution-closed in (μ, Σ).
$X \sim \text{Multinomial}_k(r, p)$, with $\text{E}[X] = rp$ and $\text{Var}(X) = r\left(\text{diag}(p) - pp^T\right)$.	Convolution-closed in r if p is fixed.

function [Jørgensen, 1992]. Furthermore, it turns out that $G_{\lambda_1,\lambda_2,x}$ has a simple closed form for several of the well-known distributions from Table 5.1; see Table 5.2. For example, if F_λ is the Poisson(λ) distribution, then $G_{\lambda_1,\lambda_2,x}$ is the Binomial $(x, \lambda_1/(\lambda_1+\lambda_2))$ distribution.

5.2.2 Data thinning

Recall from Section 5.2.1 that $G_{\lambda_1,\lambda_2,x}$ is the conditional distribution of $X' \mid X' + X'' = x$, where $X' \sim F_{\lambda_1}$ and $X'' \sim F_{\lambda_2}$ with $X' \perp\!\!\!\perp X''$. We now introduce our proposal.

Algorithm 5.1 (Data thinning) *Observe a realization x of $X \sim F_\lambda$, where the distribution F_λ is convolution-closed in λ with parameter space Λ. For any value of $\epsilon \in (0,1)$ such that $\epsilon\lambda \in \Lambda$ and $(1-\epsilon)\lambda \in \Lambda$, first draw $X^{(1)} \mid X = x \sim G_{\epsilon\lambda,(1-\epsilon)\lambda,x}$, and then let $X^{(2)} = X - X^{(1)}$.*

We now introduce our main theorem, which is motivated by a proposal by Joe [1996] to construct autoregressive time series processes with known marginal distributions.

Theorem 5.1 *Suppose that we apply Algorithm 5.1 to a realization x of $X \sim F_\lambda$. Then, the following results hold: (i) $X^{(1)} \sim F_{\epsilon\lambda}$ and $X^{(2)} \sim F_{(1-\epsilon)\lambda}$; (ii) $X^{(1)} \perp\!\!\!\perp X^{(2)}$; (iii) If F_λ*

satisfies the linear expectation property (Remark 5.1), then $E[X^{(1)}] = \epsilon\, E[X]$ *and* $E[X^{(2)}] = (1-\epsilon)\, E[X]$.

Theorem 5.1 is proven in Section D.2.1 of the supplementary materials. The intuition for parts (i) and (ii) is as follows: if $X \sim F_\lambda$, then X could have arisen as the sum of two independent random variables $X' \sim F_{\lambda_1}$ and $X'' \sim F_{\lambda_2}$, with $\lambda_1 + \lambda_2 = \lambda$. Algorithm 5.1 works backwards to undo this sum by generating $X^{(1)}$ and $X^{(2)}$ that follow the same distribution as X' and X''. Part (iii) follows from Remark 5.1. As we will see in Section 5.2.3, $\epsilon \in (0,1)$ is a tuning parameter that governs a tradeoff between how much information is in $X^{(1)}$ as opposed to $X^{(2)}$.

Theorem 5.1 guarantees that the decomposition provided by Algorithm 5.1 satisfies the goals given in Section 5.1: namely $X = X^{(1)} + X^{(2)}$, $X^{(1)} \perp\!\!\!\perp X^{(2)}$, and $X^{(1)}$ and $X^{(2)}$ follow the same distribution as X, up to a (known) scaling of a parameter. Table 5.2 summarizes the data thinning proposal for several well-known distributions. The proposal in this chapter extends well beyond this set of distributions, although in some cases the conditional distribution $G_{\lambda_1,\lambda_2,x}$ may not have a recognizable form.

Remark 5.2 *Some of the decompositions presented in Table 5.2 require knowledge of an additional parameter that is not of primary interest. For example, like the decomposition of the Gaussian given in Leiner et al. [2022] and* **?**, *thinning the* $\mathrm{N}(\mu, \sigma^2)$ *distribution requires knowledge of* σ^2. *In Section 5.2.4, we explore the implications of performing data thinning in the presence of an unknown nuisance parameter.*

Remark 5.3 *Table 5.2 indicates that thinning the* $\mathrm{Binomial}(r, p)$ *distribution requires that* ϵr *take on an integer value. This is because the binomial distribution is not infinitely divisible [Joe, 1996]. This restriction becomes more limiting in the extension to multiple folds given in Section 5.3, and prevents us from thinning the Bernoulli distribution.*

We now give an example of an application where data thinning is useful in practice.

Example 5.2 (Model evaluation for unsupervised learning using data thinning) *Suppose we observe X_{ij} for $i = 1, \ldots, n$ and $j = 1, \ldots, d$, where either each X_{ij} is drawn independently from a univariate convolution-closed distribution that satisfies the linear expectation property from Remark 5.1, or else each row $(X_{i1}, \ldots, X_{id})^T$ is drawn independently from a multivariate convolution-closed distribution that satisfies the linear expectation property.*

We wish to evaluate $\hat{\mu}(X)$ obtained from unsupervised learning (e.g. clustering) on X, as an estimator for $E[X]$. Computing a loss function between $\hat{\mu}(X)$ and X is unsatisfactory, since the loss function will take on a small value if we overfit the mean. Instead, we apply Algorithm 5.1 with $\epsilon \in (0, 1)$ to either each element or each row in X, such that each element X_{ij} is thinned into $X^{(1)}_{ij}$ and $X^{(2)}_{ij}$. We compute $\hat{\mu}(X^{(1)})$ by applying unsupervised learning to $X^{(1)}$ (step 1). This provides an estimator of $E[X^{(1)}] = \epsilon\, E[X]$ (Theorem 5.1, part (iii)). We then compute a loss function between $\hat{\mu}(X^{(1)})$ and $X^{(2)}$ (step 2). The independence between $X^{(1)}$ and $X^{(2)}$ prevents the loss function from taking on a small value due to overfitting.

In Example 5.2, if $\epsilon = 0.5$, then $\mathrm{E}[X^{(1)}] = \mathrm{E}[X^{(2)}] = 0.5\,\mathrm{E}[X]$. Thus, $\hat{\mu}(X^{(1)})$ is an estimator of $\mathrm{E}[X^{(2)}]$, and so devising a suitable loss function in step 2 is straightforward. If $\epsilon \neq 0.5$, then $\hat{\mu}(X^{(1)})$ is a plug-in estimator of $\epsilon/(1-\epsilon)\,\mathrm{E}[X^{(2)}]$ (Theorem 5.1); we discuss this further in Section 5.2.3.

5.2.3 Role of the parameter ϵ

In Algorithm 5.1, the parameter ϵ governs a trade-off between the information in $X^{(1)}$ and $X^{(2)}$.

Example 5.3 (Fisher information in thinned Poisson distribution) *Let $X \sim \mathrm{Poisson}(\lambda)$. We thin X to obtain $X^{(1)} \sim \mathrm{Poisson}(\epsilon\lambda)$ and $X^{(2)} \sim \mathrm{Poisson}((1-\epsilon)\lambda)$. Let $I_X(\lambda)$ denote the Fisher information contained in X about the parameter λ. Then $I_X(\lambda) = 1/\lambda$, $I_{X^{(1)}}(\lambda) = \epsilon I_X(\lambda)$, and $I_{X^{(2)}}(\lambda) = (1-\epsilon) I_X(\lambda)$.*

Example 5.4 (Fisher information in thinned binomial distribution) *Let $X \sim \mathrm{Binomial}(r, p)$. We thin X to obtain $X^{(1)} \sim \mathrm{Binomial}(\epsilon r, p)$ and $X^{(2)} \sim \mathrm{Binomial}((1-\epsilon)r, p)$. Let $I_X(p)$*

Table 5.2: Details of data thinning for several well-known distributions, using the parameterizations given in Table 5.1. While the exponential distribution itself is not convolution-closed, writing the Exponential(λ) distribution as Gamma($1, \lambda$) yields a decomposition. In all cases, the distribution of $X^{(2)}$ matches that of $X^{(1)}$, with ϵ replaced by $(1-\epsilon)$, with $X^{(1)} \perp\!\!\!\perp X^{(2)}$

Distribution of X	Generate $X^{(1)} \mid X = x$ as:	Dist. of $X^{(1)}$	Notes
Poisson(λ)	Draw $X^{(1)} \mid X = x \sim \text{Binomial}(x, \epsilon)$.	Poisson($\epsilon\lambda$)	
N(μ, σ^2)	Draw $X^{(1)} \mid X = x \sim \text{N}(\epsilon x, \epsilon(1-\epsilon)\sigma^2)$.	N($\epsilon\mu, \epsilon\sigma^2$)	σ^2 must be known.
NegativeBinomial(r, p)	Draw $X^{(1)} \mid X = x \sim \text{BetaBinomial}(x, \epsilon r, (1-\epsilon)r)$.	NegativeBinomial($\epsilon r, p$)	r must be known.
Gamma(α, β)	Draw $Z \sim \text{Beta}(\epsilon\alpha, (1-\epsilon)\alpha)$, and let $X^{(1)} = x \cdot Z$.	Gamma($\epsilon\alpha, \beta$)	α must be known.
Exponential(λ)	Draw $Z \sim \text{Beta}(\epsilon, (1-\epsilon))$, and let $X^{(1)} = x \cdot Z$.	Gamma(ϵ, λ)	
Binomial(r, p)	Draw $X^{(1)} \mid X = x \sim \text{Hypergeometric}(\epsilon r, (1-\epsilon)r, x)$.	Binomial($\epsilon r, p$)	r must be known ϵr must be integer.
$\text{N}_k(\mu, \Sigma)$	Draw $X^{(1)} \mid X = x \sim \text{N}(\epsilon x, \epsilon(1-\epsilon)\Sigma)$.	$\text{N}_k(\epsilon\mu, \epsilon\Sigma)$	Σ must be known.
$\text{Multinomial}_k(r, p)$	Draw $X^{(1)} \mid X = x \sim$ $\text{MultivariateHypergeometric}(x_1, x_2, \ldots, x_k, \epsilon r)$.	$\text{Multinomial}_k(\epsilon r, p)$	r must be known. ϵr must be integer.

denote the Fisher information contained in X about the parameter p. Then $I_X(p) = r/(p(1-p))$, $I_{X^{(1)}}(p) = \epsilon I_X(p)$, and $I_{X^{(2)}}(p) = (1-\epsilon)I_X(p)$.

Similar results hold for other distributions in Table 5.2. Intuitively, as ϵ increases, the amount of information in $X^{(1)}$ about the parameter of interest increases, and the amount of information in $X^{(2)}$ decreases. This has implications for Example 5.2: as ϵ increases, the quality of the estimator of the expected value increases (step 1), but the information available for computing the loss between this estimator and $X^{(2)}$ decreases (step 2).

In Section 5.2.2, we mentioned that the loss function in step 2 of Example 5.2 must be chosen with care when $\epsilon \neq 0.5$. This can be seen in the following example.

Example 5.5 (Example 5.2 with mean squared error loss) *Consider step 2 of Example 5.2 with mean squared error loss. Since $E[X^{(2)}] = (1-\epsilon)/\epsilon \times E[X^{(1)}]$, we compute the loss as*

$$\frac{1}{nd}\left\| X^{(2)} - \frac{1-\epsilon}{\epsilon}\hat{\mu}(X^{(1)}) \right\|_F^2,$$

where the factor of $(1-\epsilon)/\epsilon$ turns an estimate of $E[X^{(1)}]$ into an estimate of $E[X^{(2)}]$.

As we will see in Section 5.4, we can also use alternative loss functions, such as negative log likelihood, when considering Example 5.2.

Example 5.2 focuses on using data thinning to evaluate an estimator, and Sections 5.4 and 5.5 of this chapter focus on model selection and assessment. However, data thinning can also be applied in a variety of other settings, such as inference after variable selection in regression [Leiner et al., 2022]. In these settings, the parameter ϵ still governs a trade-off between the information available in $X^{(1)}$, used for selection, and $X^{(2)}$, used for inference.

5.2.4 Effect of unknown nuisance parameters

For several of the distributions in Table 5.2, data thinning requires knowledge of a nuisance parameter. For example, thinning a $\mathrm{N}(\mu, \sigma^2)$ distribution requires knowledge of σ^2.

We now consider what happens when we perform data thinning on normally-distributed data using an incorrect value of the variance. We refer to this incorrect value as $\tilde{\sigma}^2$.

Proposition 5.2.1 *Suppose that we observe x from $X \sim \mathrm{N}(\mu, \sigma^2)$. We draw $X^{(1)} \mid X = x \sim \mathrm{N}\left(\epsilon x, \epsilon(1-\epsilon)\tilde{\sigma}^2\right)$, for some $\tilde{\sigma}$ that is not a function of x, and let $X^{(2)} = X - X^{(1)}$. Then: (i) $X^{(1)} \sim \mathrm{N}\left(\epsilon\mu, \epsilon^2\sigma^2 + \epsilon(1-\epsilon)\tilde{\sigma}^2\right)$, (ii) $X^{(2)} \sim \mathrm{N}\left((1-\epsilon)\mu, (1-\epsilon)^2\sigma^2 + \epsilon(1-\epsilon)\tilde{\sigma}^2\right)$, and (iii)* $\mathrm{cov}\left(X^{(1)}, X^{(2)}\right) = \epsilon(1-\epsilon)\left(\sigma^2 - \tilde{\sigma}^2\right)$.

Proposition 5.2.1(iii) indicates that if we apply data thinning with not enough noise ($\tilde{\sigma}^2 < \sigma^2$), then $X^{(1)}$ and $X^{(2)}$ are positively correlated. On the other hand, if we apply data thinning with too much noise ($\tilde{\sigma}^2 > \sigma^2$), then $X^{(1)}$ and $X^{(2)}$ are negatively correlated. Similar results hold for the negative binomial distribution and the Gamma distribution.

Proposition 5.2.2 *Suppose that we observe x from $X \sim \mathrm{NegativeBinomial}(r, p)$. We draw $X^{(1)} \mid X = x \sim \mathrm{BetaBin}\left(x, \epsilon\tilde{r}, (1-\epsilon)\tilde{r}\right)$ for some $\tilde{r}$ that is not a function of x, and let $X^{(2)} = X - X^{(1)}$. Then* $\mathrm{cov}\left(X^{(1)}, X^{(2)}\right) = \epsilon(1-\epsilon)r\left(\frac{1-p}{p}\right)^2\left(1 - \frac{r+1}{\tilde{r}+1}\right)$.

Proposition 5.2.3 *Suppose that we observe x from $X \sim \mathrm{Gamma}(\alpha, \beta)$. We let $X^{(1)} = x \times Z$, where $Z \sim \mathrm{Beta}\left(\epsilon\tilde{\alpha}, (1-\epsilon)\tilde{\alpha}\right)$ for some $\tilde{\alpha}$ that is not a function of x. We let $X^{(2)} = X - X^{(1)}$. Then* $\mathrm{cov}\left(X^{(1)}, X^{(2)}\right) = \epsilon(1-\epsilon)\frac{\alpha}{\beta^2}\left(1 - \frac{\alpha+1}{\tilde{\alpha}+1}\right)$.

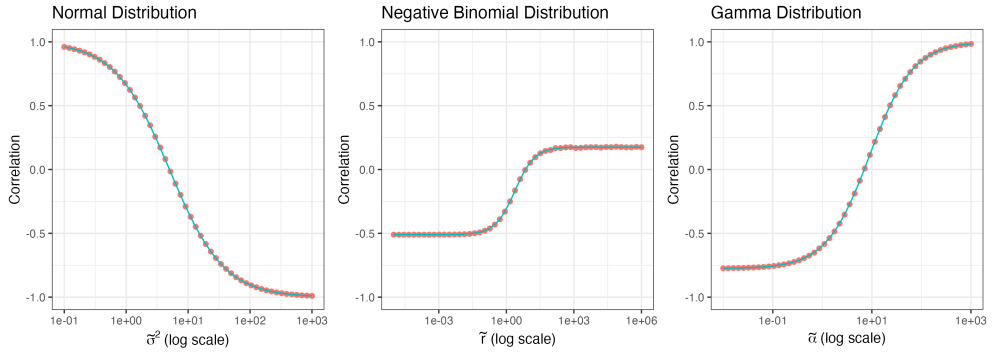


Figure 5.1: *Left:* We generate 100,000 realizations of $X \sim \mathrm{N}(7, 5)$. For 50 values of $\tilde{\sigma}^2$, we thin X into $X^{(1)}$ and $X^{(2)}$ using $\tilde{\sigma}^2$ instead of $\sigma^2 = 5$. *Center:* We generate 100,000 realizations of $X \sim \mathrm{NB}(7, 0.7)$. For 50 values of $\tilde{r}$, we thin X into $X^{(1)}$ and $X^{(2)}$ using $\tilde{r}$ instead of $r = 7$. *Right:* We generate 100,000 realizations of $X \sim \mathrm{Gamma}(7, 5)$. For 50 values of $\tilde{\alpha}$, we thin X into $X^{(1)}$ and $X^{(2)}$ using $\tilde{\alpha}$ instead of $\alpha = 7$. *All:* In each panel, for each value of the nuisance parameter, we display the empirical correlation between $X^{(1)}$ and $X^{(2)}$ (red dots), along with the theoretical correlation suggested by Propositions 5.2.1–5.2.3 (blue lines).

Propositions 5.2.1–5.2.3 are proven in Section D.2.2. Figure 5.1 verifies these results empirically. The results in this section assume that $\tilde{\sigma}$, $\tilde{b}$, and $\tilde{\alpha}$ are not a function of X. In practice, one might estimate the unknown parameters σ, b, and α using additional data.

5.3 Multifold data thinning

Data thinning involves decomposing X into $X^{(1)}$ and $X^{(2)}$, which each have the same distribution as X (up to a known parameter scaling). It can be applied recursively to create M independent data folds, $X^{(1)}, \ldots, X^{(M)}$, that sum to X, as in the following example.

Example 5.6 (Recursive thinning of the normal distribution) *Let x denote a realization of $X \sim N(\mu, \sigma^2)$. Given $\epsilon_1, \epsilon_2, \epsilon_3 \in (0, 1)$ with $\epsilon_1 + \epsilon_2 + \epsilon_3 = 1$, we first draw $X^{(1)} \mid X \sim \mathrm{N}\left(\epsilon_1 X, \epsilon_1(1-\epsilon_1)\sigma^2\right)$. Let $X^{(2,3)} = X - X^{(1)}$. By Theorem 5.1, $\left(X^{(1)}, X^{(2,3)}\right) \sim N\left(\epsilon_1\mu, \epsilon_1\sigma^2\right) \times N\left((1-\epsilon_1)\mu, (1-\epsilon_1)\sigma^2\right)$.*

We next draw $X^{(2)} \mid X^{(2,3)} \sim \mathrm{N}\left(\frac{\epsilon_2}{1-\epsilon_1}X^{(2,3)}, \frac{\epsilon_2}{1-\epsilon_1}(1-\frac{\epsilon_2}{1-\epsilon_1})(1-\epsilon_1)\sigma^2\right)$, and let $X^{(3)} =$

$X - X^{(1)} - X^{(2)}$. *By Theorem 5.1,* $\left(X^{(2)}, X^{(3)}\right) \sim N(\epsilon_2\mu, \epsilon_2\sigma^2) \times N(\epsilon_3\mu, \epsilon_3\sigma^2)$. *Furthermore, since* $\left(X^{(2)}, X^{(3)}\right)$ *is a function of* $X^{(2,3)}$*, each of* $X^{(2)}$ *and* $X^{(3)}$ *remain independent of* $X^{(1)}$. *Thus,* $\left(X^{(1)}, X^{(2)}, X^{(3)}\right) \sim N(\epsilon_1\mu, \epsilon_1\sigma^2) \times N(\epsilon_2\mu, \epsilon_2\sigma^2) \times N(\epsilon_3\mu, \epsilon_3\sigma^2)$.

While Example 5.6 can be extended to create $M > 3$ folds, this recursive approach can be cumbersome. In Example 5.1 of Section 5.1, we saw that, for the Gamma distribution, there is a simple way to create multiple folds without recursion. We will now provide a general form of this result. Let $G_{\lambda_1,\lambda_2,\ldots,\lambda_M,x}$ denote the joint distribution of $(X_1, \ldots, X_M) \mid X_1 + X_2 + \ldots + X_M = x$, where $X_m \overset{\text{ind.}}{\sim} F_{\lambda_m}$, for $m = 1, \ldots, M$, and where F_λ is a convolution-closed distribution. The following algorithm and theorem mimic Algorithm 5.1 and Theorem 5.1.

Algorithm 5.2 (Multifold data thinning) *Observe a realization* x *of* $X \sim F_\lambda$*, where* F_λ *is a convolution-closed distribution with parameter space* Λ. *First, choose* $\epsilon_1, \ldots, \epsilon_M \in (0, 1)$ *such that* $\sum_{m=1}^{M} \epsilon_m = 1$ *and* $\epsilon_m\lambda \in \Lambda$ *for* $m = 1, \ldots M$. *Then, draw* $\left(X^{(1)}, \ldots, X^{(M)}\right) \sim G_{\epsilon_1\lambda,\epsilon_2\lambda,\ldots,\epsilon_M\lambda,x}$.

Theorem 5.2 *Suppose we apply Algorithm 5.2 to a realization* x *of* $X \sim F_\lambda$*, for a convolution-closed distribution* F_λ. *Then, the following results hold: (i)* $X^{(m)} \sim F_{\epsilon_m\lambda}$ *for* $m = 1, \ldots, M$*; (ii)* $X^{(1)}, \ldots, X^{(M)}$ *are mutually independent; (iii)* $X^{(1)} + X^{(2)} + \cdots + X^{(M)} = X$*; and (iv) if* F_λ *satisfies the linear expectation property (Remark 5.1), then* $E[X^{(m)}] = \epsilon_m E[X]$ *for* $m = 1, \ldots, M$.

The proof of Theorem 5.2 is included in Section D.2.1, and is a straightforward extension of that of Theorem 5.1. The intuition for parts (i)-(iii) is as follows: we know that $X \sim F_\lambda$ could have arisen as the sum of M mutually independent random variables $X_1, \ldots, X_M$ such that $X_m \sim F_{\epsilon_m\lambda}$. If we draw $\left(X^{(1)}, \ldots, X^{(M)}\right) | X = x \sim G_{\epsilon_1\lambda,\epsilon_2\lambda,\ldots,\epsilon_M\lambda,x}$, then the joint distribution of $\left(X^{(1)}, \ldots, X^{(M)}\right)$ equals the joint distribution of $(X_1, \ldots, X_M)$, i.e. it is the joint distribution of M independent random variables with distributions $F_{\epsilon_1\lambda}, \ldots, F_{\epsilon_M\lambda}$. Part (iv) follows directly from Remark 5.1. We now revisit the case of the normal distribution from Example 5.6.

Table 5.3: Details of how to perform multifold data thinning (Algorithm 5.2) for several common univariate distributions, where where $\epsilon = (\epsilon_1, \ldots, \epsilon_M)^T$. In the decomposition of the binomial distribution, $\epsilon_m r$ must be an integer. Each row can be verified using properties of these distributions

Distribution of X	Generate $\left(X^{(1)}, \ldots, X^{(M)}\right) \mid X = X$ as:	Dist. of $X^{(m)}$
Poisson(λ)	$\left(X^{(1)}, \ldots, X^{(M)}\right) \mid X = x \sim \text{Multinomial}(x, \epsilon_1, \ldots, \epsilon_M)$.	Poisson($\epsilon_m \lambda$)
N(μ, σ^2)	$\left(X^{(1)}, \ldots, X^{(M)}\right) \mid X = x \sim \text{N}\left(\mu\boldsymbol{\epsilon}, \sigma^2 \text{diag}(\epsilon) - \sigma^2 \epsilon\epsilon^T\right)$,	N($\epsilon_m \mu, \epsilon_m \sigma^2$),
NegativeBinomial(r, p)	$\left(X^{(1)}, \ldots, X^{(M)}\right) \mid X = x \sim \text{DirichletMultinomial}(X, \epsilon_1 r, \ldots, \epsilon_M r)$.	NegativeBinomial($\epsilon_m r, p$)
Gamma(α, β)	Draw $Z \sim \text{Dirichlet}\left(\epsilon_1 \alpha, \ldots, \epsilon_M \alpha\right)$, and and let $\left(X^{(1)}, \ldots, X^{(M)}\right) = x \cdot Z$	Gamma($\epsilon_m \alpha, \beta$)
Exponential(λ)	Draw $Z \sim \text{Dirichlet}(\epsilon_1, \ldots, \epsilon_M)$, and let $\left(X^{(1)}, \ldots, X^{(M)}\right) = x \cdot Z$.	Gamma(ϵ_m, λ)
Binomial(r, p)	$\left(X^{(1)}, \ldots, X^{(M)}\right) \mid X = x \sim \text{MultivariateHypergeometric}(\epsilon_1 r, \ldots, \epsilon_M r, x)$.	Binomial($\epsilon_m r, p$)

Example 5.7 (Multifold thinning of the normal distribution) *Let $X \sim N(\mu, \sigma^2)$ and let $\epsilon_1, \epsilon_2, \epsilon_3 > 0$ with $\sum_{i=1}^3 \epsilon_i = 1$. To generate $M = 3$ independent folds of the data, we draw*

$$\begin{bmatrix} X^{(1)} \\ X^{(2)} \\ X^{(3)} \end{bmatrix} \mid X = x \sim N\left(\begin{bmatrix} \epsilon_1 x \\ \epsilon_2 x \\ \epsilon_3 x \end{bmatrix}, \begin{bmatrix} \epsilon_1(1-\epsilon_1)\sigma^2 & -\epsilon_1\epsilon_2\sigma^2 & -\epsilon_1\epsilon_3\sigma^2 \\ -\epsilon_1\epsilon_2\sigma^2 & \epsilon_2(1-\epsilon_2)\sigma^2 & -\epsilon_2\epsilon_3\sigma^2 \\ -\epsilon_1\epsilon_3\sigma^2 & -\epsilon_2\epsilon_3\sigma^2 & \epsilon_3(1-\epsilon_3)\sigma^2 \end{bmatrix} \right).$$

One can verify that this multivariate normal corresponds to $G_{\epsilon_1\lambda, \epsilon_2\lambda, \epsilon_3\lambda, x}$. By Theorem 5.2, $X^{(1)}, X^{(2)}$, and $X^{(3)}$ are independent and $X^{(m)} \sim N(\epsilon_m \mu, \epsilon_m \sigma^2)$ for $m = 1, 2, 3$. This distribution $G_{\epsilon_1\lambda, \epsilon_2\lambda, \epsilon_3\lambda, x}$ is a degenerate multivariate normal distribution, which enforces the constraint that the realized values of $X^{(1)}, X^{(2)}$, and $X^{(3)}$ sum to x.

Table 5.3 reveals that $G_{\epsilon_1\lambda, \epsilon_2\lambda, \ldots, \epsilon_M\lambda, x}$ in Algorithm 5.2 has a very simple form for every univariate distribution in Table 5.2. We omit the multivariate distributions to avoid cumbersome notation.

The parameters $\epsilon_1, \ldots, \epsilon_M$ in Algorithm 5.2 govern the same information tradeoff that was described in Section 5.2.3. Folds with the largest values of ϵ_m contain the most information about the parameter of interest. We now consider the following extension of Example 5.2.

Example 5.8 (Cross validation for unsupervised learning using multifold thinning) *In the setting of Example 5.2, we apply Algorithm 5.2 with parameters $\epsilon_1, \ldots, \epsilon_M$ to thin each element X_{ij} into $X_{ij}^{(1)}, \ldots, X_{ij}^{(M)}$.*

Then, for $m = 1, \ldots, M$, we first define $X^{(-m)} := X - X^{(m)}$, and apply unsupervised learning to $X^{(-m)}$ to obtain $\hat{\mu}\left(X^{(-m)}\right)$, which is an estimator of $E[X^{(-m)}] = (1 - \epsilon_m)\, E[X]$ (step 1). We then compute a loss function between $\hat{\mu}\left(X^{(-m)}\right)$ and $X^{(m)}$ (step 2). For example, as in Example 5.5, we can compute the mean squared error between $\frac{\epsilon_m}{1-\epsilon_m}\hat{\mu}(X^{(-m)})$ and $X^{(m)}$. We evaluate the estimator $\hat{\mu}(\cdot)$ by averaging the loss across folds.

For simplicity, we suggest setting $\epsilon_1 = \ldots = \epsilon_M = 1/M$. The advantage of multifold thinning (Example 5.8) over single fold thinning with $\epsilon = 1/M$ (Example 5.2) is reduction of the variance of the loss function via averaging. We will demonstrate the practical advantages of multifold thinning in Section 5.4.

5.4 Simulation study

5.4.1 Simulation setup

Data thinning can be applied to a variety of settings, including the inference after model selection problems considered in Leiner et al. [2022], and to cross validation for supervised learning. Here, we focus on the application of data thinning to cross-validation for unsupervised learning. Data thinning is particularly attractive in this setting, as a traditional sample splitting approach is not suitable; see Owen and Perry [2009] or Fu and Perry [2020] for further discussion. In what follows, we apply data thinning and multifold thinning to unsupervised learning problems, and contrast their performance against naive approaches that use the same data to both fit and validate the unsupervised models. Specifically, we consider Examples 5.9 and 5.10.

Example 5.9 (Choosing the number of principal components on binomial data) *We generate data with $n = 250$ observations and $d = 100$ dimensions. Specifically, for $i = 1, \ldots, n$ and $j = 1, \ldots, d$, we generate $X_{ij} \overset{\text{ind.}}{\sim} Binomial(r, p_{ij})$ where $r = 100$ and p is an unknown $n \times d$ matrix of probabilities. We construct logit(p) as a rank-$K^* = 10$ matrix with singular values $5, 6, \ldots, 14$. Additional details are provided in Section D.4. Our goal is to estimate K^*.*

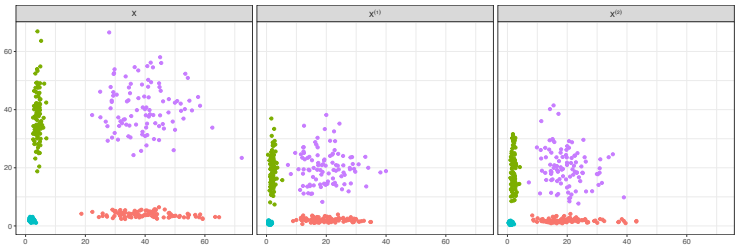


Figure 5.2: *Left*: A simulated dataset in the $d = 2$, $K^* = 4$ setting described in Example 5.10. *Center/Right*: The result of data thinning with $\epsilon = 0.5$.

Example 5.10 (Choosing the number of clusters on Gamma data) *We generate datasets* $X \in \mathbb{R}^{n \times d}$ *such that there are 100 observations in each of* K^* *clusters, for a total of* $n = 100K^*$ *observations. Our objective is to estimate* K^**. We let* $X_{ij} \overset{\text{ind.}}{\sim} Gamma(\lambda, \theta_{c_i,j})$*, for* $i = 1, \ldots, n$ *and* $j = 1, \ldots, d$*, where* $c_i \in \{1, 2, \ldots, K^*\}$ *indexes the true cluster membership of the* i*th observation. The shape parameter* λ *is a known constant common across all clusters and all dimensions, whereas the rate parameter* θ *is an unknown* $K^* \times d$ *matrix such that each cluster has its own* d*-dimensional rate parameter. We generate data under two regimes: (1) a small* d*, small* K^* *regime in which* $d = 2$ *and* $K^* = 4$*, and (2) a large* d*, large* K^* *regime in which* $d = 100$ *and* $K^* = 10$*. The values of* λ *and* θ *are provided in Section D.4. A sample "small* d*, small* K^**" dataset is presented in Figure 5.2, alongside the output of data thinning with* $\epsilon = 0.5$*.*

5.4.2 Methods

We use Algorithm 5.3 to select the number of principal components in binomial data, as in Example 5.9, using data thinning.

Algorithm 5.3 (Evaluating binomial principal components with negative log-likelihood loss) *Input: A positive integer* K*, a matrix* $X \in \mathbb{Z}_{\geq 0}^{n \times d}$*, where* $X_{ij} \overset{\text{ind.}}{\sim} \text{Binomial}(r, p_{ij})$*, and non-negative scalars* $\epsilon^{(\text{train})}$ *and* $\epsilon^{(\text{test})} = 1 - \epsilon^{(\text{train})}$ *such that* $\epsilon^{(\text{train})} r, \epsilon^{(\text{test})} r \in \mathbb{Z}_{>0}$*.*

1. *Apply data thinning to X to obtain $X^{(\text{train})}$ and $X^{(\text{test})}$, where $X_{ij}^{(\text{train})} \overset{\text{ind.}}{\sim} \text{Binomial}\left(\epsilon^{(\text{train})} r, p_{ij}\right)$ and $X_{ij}^{(\text{test})} \overset{\text{ind.}}{\sim} \text{Binomial}\left(\epsilon^{(\text{test})} r, p_{ij}\right)$*

2. *Compute the singular value decomposition of the log-odds of $X^{(\text{train})}$, $\text{logit}\left\{(X_{ij}^{(\text{train})} + 0.001)/(\epsilon^{(\text{train})} r + 0.002)\right\} = \hat{U}\hat{D}\hat{V}^T$. Pseudo-counts prevent taking the logit of 0 or 1.*

3. *Construct the rank-K approximation of $X^{(\text{train})}$, $p^{(K)} = \text{expit}\left(\hat{U}_{1:K}\hat{D}_{1:K}\hat{V}_{1:K}\right)$.*

4. *Compute the negative log-likelihood loss on $X^{(\text{test})}$ as $-\sum_{i=1}^{n}\sum_{j=1}^{d} \log f\left(X_{ij}^{(\text{test})} \middle| \epsilon^{(\text{test})} r, p_{ij}^{(K)}\right)$, where $f(\cdot \mid r, p)$ is the density function for the $\text{Binomial}(r, p)$ distribution.*

We use Algorithm 5.4 to select the number of clusters in Gamma data, as in Example 5.10, using data thinning.

Algorithm 5.4 (Evaluating Gamma clusters with negative log-likelihood loss) *Input: A positive integer K, and $X \in \mathbb{R}^{n \times d}$ where $X_{ij} \overset{\text{ind.}}{\sim} \text{Gamma}\left(\lambda, \theta_{c_i,j}\right)$. Here, $\theta \in (0,\infty)^{K^* \times d}$ where $\theta_{c_i,j}$ is the true but unknown rate parameter for the c_ith cluster in the jth dimension, $c_i \in \{1, 2, \ldots, K^*\}$, and λ is the known shape parameter. Also, non-negative scalars $\epsilon^{(\text{train})}$ and $\epsilon^{(\text{test})} = 1 - \epsilon^{(\text{train})}$.*

1. *Apply data thinning to X to obtain $X^{(\text{train})}$ and $X^{(\text{test})}$, where $X_{ij}^{(\text{train})} \overset{\text{ind.}}{\sim} \text{Gamma}\left(\epsilon^{(\text{train})}\lambda, \theta_{c_i,j}\right)$ and $X_{ij}^{(\text{test})} \overset{\text{ind.}}{\sim} \text{Gamma}\left(\epsilon^{(\text{test})}\lambda, \theta_{c_i,j}\right)$.*

2. *Run K-means on $X^{(\text{train})}$ to estimate K clusters. Denote the cluster assignment of the ith observation as $\hat{c}_i$.*

3. *Within each cluster, estimate the parameters using $X^{(\text{train})}$ [Ye and Chen, 2017, Louzada et al., 2019]. Let $\hat{\lambda}^{(K)}$ and $\hat{\theta}^{(K)}$ denote the $K \times d$ estimated parameter matrices.*

4. *Compute the loss on $X^{(\text{test})}$ as $-\sum_{i=1}^{n}\sum_{j=1}^{d} \log f\left(X_{ij}^{(\text{test})} \middle| \hat{\lambda}_{\hat{c}_i,j}^{(K)} \epsilon^{(\text{test})}/\epsilon^{(\text{train})}, \hat{\theta}_{\hat{c}_i,j}^{(K)}\right)$, where $f(\cdot \mid \lambda, \theta)$ is the density function for the $\text{Gamma}(\lambda, \theta)$ distribution.*

We apply Algorithms 5.3 and 5.4 in three different ways. First, we apply them without modification, with $\epsilon^{(\text{train})} = 0.5$ and $\epsilon^{(\text{train})} = 0.8$. Next, we slightly modify these algorithms by replacing step 1 with multi-fold thinning (Algorithm 5.2) with $M = 5$ and $\epsilon_1 = \cdots = \epsilon_M = 0.2$. For $m = 1, \ldots, M$, we then perform steps 2–4 using $X^{(\text{train})} = X - X^{(m)}$, $\epsilon^{(\text{train})} = (M-1)/M$ and $X^{(\text{test})} = X^{(m)}$, $\epsilon^{(\text{test})} = 1/M$. We then average the loss functions obtained across the M applications of step 4. Finally, we consider a naive method that re-uses data, by skipping step 1, and simply taking $X^{(\text{train})} = X^{(\text{test})} = X$ in steps 2–4 and $\epsilon^{(\text{train})} = \epsilon^{(\text{test})} = 1$ in step 4.

Our goal is to select the value of K that minimizes the loss function. Because data thinning produces independent training and test sets, we expect that the data thinning approaches will produce U-shaped loss function curves, as a function of K. By contrast, in the naive approach, the full data X is used to fit the model and to compute the loss functions in Algorithms 5.3 and 5.4, resulting in monotonically decreasing loss curves, as a function of K.

Other loss functions can be used in lieu of the negative log-likelihood loss in Algorithms 5.3 and 5.4. In Section D.5 of the supplementary materials, we extend Algorithms 5.3 and 5.4 to the case of mean squared error loss, and show similar results.

5.4.3 Results

Figure 5.3 displays the loss function for all three simulation settings as a function of K; results have been averaged over $2,000$ simulated datasets and rescaled to the $[0, 1]$ interval for ease of comparison. The values of K with the lowest average loss function are circled on the plots. As expected, the data thinning approaches in Figure 5.3 exhibit sharp minimum values, as opposed to the monotonically decreasing curves produced by the naive method. The data thinning approaches correctly select the true value of $K = K^*$ in all three settings, except for data thinning with $\epsilon^{(\text{train})} = 0.5$ in the binomial principal components setting. In that case, the low value of $\epsilon^{(\text{train})}$ allocates too much information to the test set, resulting in inadequate signal from the weakest principal components in the training set. Selecting a

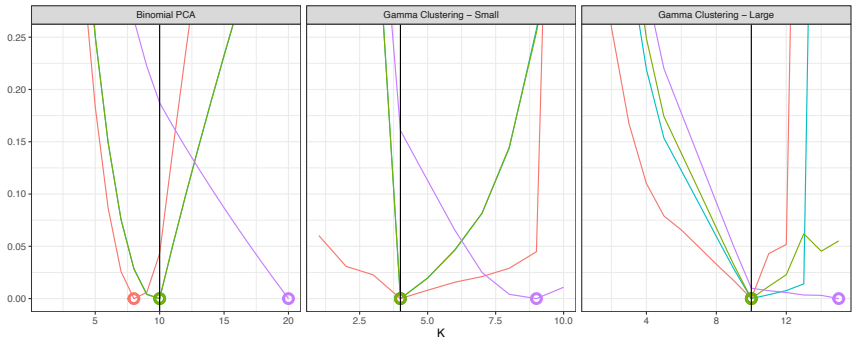


Figure 5.3: The negative log-likelihood loss averaged over 2,000 simulated data sets, as a function of K, for the naive method (purple), data thinning with $\epsilon^{(\text{train})} = 0.5$ (red), data thinning with $\epsilon^{(\text{train})} = 0.8$ (blue), and multifold thinning with $M = 5$ folds (green). Each curve has been rescaled to take on values between 0 and 1, for ease of comparison. The minimum loss values for each method are circled, and K^* is indicated by the vertical black line.

larger value of $\epsilon^{(\text{train})}$ remedies this issue, as seen with $\epsilon^{(\text{train})} = 0.8$.

We further investigate the role of $\epsilon^{(\text{train})}$ by repeating the simulation study using different values of $\epsilon^{(\text{train})}$ for single-fold data thinning. In Figure 5.4, we plot the proportion of simulations that select the correct value of K^* (i.e. the proportion of simulations in which the loss function is minimized at $K = K^*$) in each of the three settings, as a function of $\epsilon^{(\text{train})}$. We find that in the Gamma clustering simulations, lower values of ϵ are adequate. However, settings with weaker signal, such as the binomial principal components example, require larger values of ϵ to identify the true latent structure. In all settings, as ϵ approaches 1, performance begins to decay. This is a consequence of inadequate information remaining in the test set under large values of ϵ, and is consistent with the discussion of Section 5.2.3. These findings suggest that in practice, the optimal value of $\epsilon^{(\text{train})}$ is context-dependent.

Finally, we examine the benefits of multifold data thinning over single-fold data thinning. Figure 5.5 displays histograms of the number of simulations that select each value of K. Here we only include data thinning with $\epsilon^{(\text{train})} = 0.8$ and multifold thinning with $M = 5$, so that both methods use the same allocation of information between training and test sets. We see

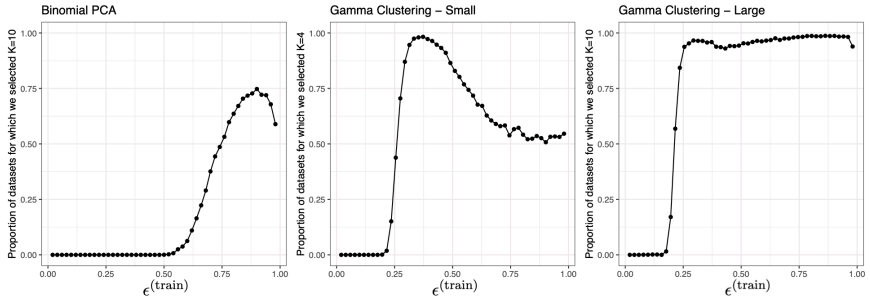


Figure 5.4: The proportion of simulations for which data thinning selects the true value of K^* with the negative log-likelihood loss, as a function of $\epsilon^{(\text{train})}$, for the simulation study described in Section 5.4.1. The optimal value of $\epsilon^{(\text{train})}$ depends on the problem at hand.

that multifold thinning generally selects the correct value of K more often than single-fold data thinning, mirroring the improvement of M-fold cross-validation using sample splitting over single-fold sample splitting in supervised settings. However, in the large Gamma setting, the signal is strong enough that multifold thinning does not provide a benefit over single-fold thinning.

5.5 Selecting the number of principal components in gene expression data

In this section, we revisit an analysis of a dataset from a single-cell RNA sequencing experiment conducted on a set of peripheral blood mononuclear cells. The dataset is freely available from 10X Genomics, and was previously analyzed in the "Guided Clustering Tutorial" vignette [Hoffman et al., 2022] for the popular `R` package `Seurat` [Hao et al., 2021, Stuart et al., 2019, Satija et al., 2015].

The dataset X is a sparse matrix of non-negative integers, representing counts from $32,738$ genes in each of $2,700$ cells. We consider applying principal components analysis to learn a low-dimensional representation of the data. In the `Seurat` vignette, filtering, normalization, log-transformation, feature selection, centering, and scaling are applied to the data, yielding a transformed matrix $\tilde{Y} \in \mathbb{R}^{2638 \times 2000}$. Details are provided in Section D.6 of the supplementary materials. Finally, the singular value decomposition of $\tilde{Y}$ is computed,

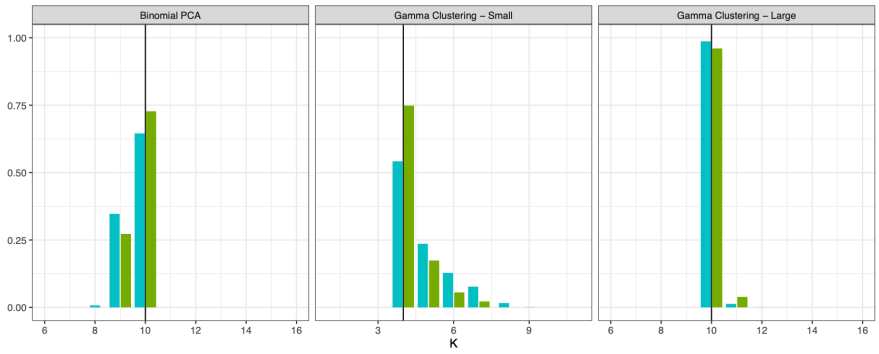


Figure 5.5: The proportion of simulated data sets in which each candidate value of K is selected, with the negative log-likelihood loss, under data thinning with $\epsilon^{(\text{train})} = 0.8$ (blue) and multifold thinning with $M = 5$ (green), for each of the simulation settings described in Section 5.4.1. The true value of K^* is indicated by the vertical black line. Multifold thinning tends to select the true value of K more often than single-fold thinning.

such that $\tilde{Y} = UDV^T$. Here we let U_k represent the kth column of the matrix U, and let $U_{1:K}D_{1:K}V_{1:K}^T$ represent the rank-K approximation of $\tilde{Y}$.

Our goal is to select the number of dimensions to use in this low-rank approximation. In the `Seurat` vignette, the authors rely on heuristic solutions such as looking for an elbow in the plot of the standard deviation of $U_K D_K$ as a function of K; see Figure 5.6(a) [James et al., 2013]. Based on the elbow plot, the authors suggest retaining around 7 principal components. Other heuristic approaches suggest as many as 12 principal components.

Before introducing the data thinning solution, we introduce a squared-error based formulation that is mathematically equivalent to the traditional elbow plot (see Section D.6), but will facilitate a direct comparison with data thinning. For $K = 1, \ldots, 20$, we compute the sum of squared errors between the matrix $\tilde{Y}$ and its rank-K approximation:

$$\left\| \tilde{Y} - U_{1:K} D_{1:K} V_{1:K}^T \right\|_F^2 .$$

Because the low-rank approximation $U_{1:K}D_{1:K}V_{1:K}^T$ is computed using $\tilde{Y}$, this loss function monotonically decreases with K. A heuristic solution for deciding how many principal com-

ponents to retain involves looking for the point in which the slope of the curve in Figure 5.6(b) begins to flatten. While this appears to happen around 5–7 principal components, which is consistent with the finding from Figure 5.6(a), the exact number of principal components to retain is still unclear. We now show that data thinning provides a less heuristic approach for estimating the number of principal components.

Single-cell RNA-sequencing data are often modeled as independent Poisson random variables [Wang et al., 2018, Sarkar and Stephens, 2021]. Thus, we assume that $X_{ij} \sim \text{Poisson}(\Lambda_{ij})$. Starting with the raw data matrix $X \in \mathbb{Z}_{\geq 0}^{2700 \times 32738}$, we perform Poisson data thinning with $\epsilon = 0.5$ to obtain a training set $X^{(1)}$ and a test set $X^{(2)}$, which are independent if the Poisson assumption holds. Furthermore, as $\epsilon = 0.5$, they are identically distributed. We then carry out the data processing described in Section D.6 on $X^{(1)}$ to obtain $\tilde{Y}^{(1)} \in \mathbb{R}^{2638 \times 2000}$. We obtain $\tilde{Y}^{(2)} \in \mathbb{R}^{2638 \times 2000}$ by applying the same data processing steps to $X^{(2)}$, but retaining only the features that were selected on $X^{(1)}$, so that the rows and columns of $\tilde{Y}^{(1)}$ and $\tilde{Y}^{(2)}$ correspond to the same genes and cells. Details are in Section D.6.

We compute the singular value decomposition on the training set, $\tilde{Y}^{(1)} = U^{(1)} D^{(1)} (V^{(1)})^T$. For a range of values of K, we then compute the sum of squared errors between $\tilde{Y}^{(2)}$ and $U^{(1)}_{1:K} D^{(1)}_{1:K} (V^{(1)}_{1:K})^T$:

$$\left\| \tilde{Y}^{(2)} - U^{(1)}_{1:K} D^{(1)}_{1:K} (V^{(1)}_{1:K})^T \right\|_F^2. \tag{5.1}$$

The results are shown in Figure 5.6(c). As we are not computing and evaluating the singular value decomposition using the same data, the plot of K vs. the loss function is not monotonically decreasing in K. Instead, it reaches a clear minimum at $K = 7$, suggesting that the rank-7 approximation provides the best fit to the observed data. Thus, data thinning provides a simple and non-heuristic way to select the number of principal components.

Remark 5.4 (Choice of ϵ) *If we had chosen $\epsilon \neq 0.5$, then while $E[X^{(2)}] = (1-\epsilon)/\epsilon \, E[X^{(1)}]$, the relationship between $E[\tilde{Y}^{(1)}]$ and $E[\tilde{Y}^{(2)}]$ would depend on the details of the data processing described in Section 5.2, and the loss function in* (5.1) *would need to be modified accordingly.*

Remark 5.5 (Overdispersion) *While we used a Poisson model for scRNA-seq data, there*

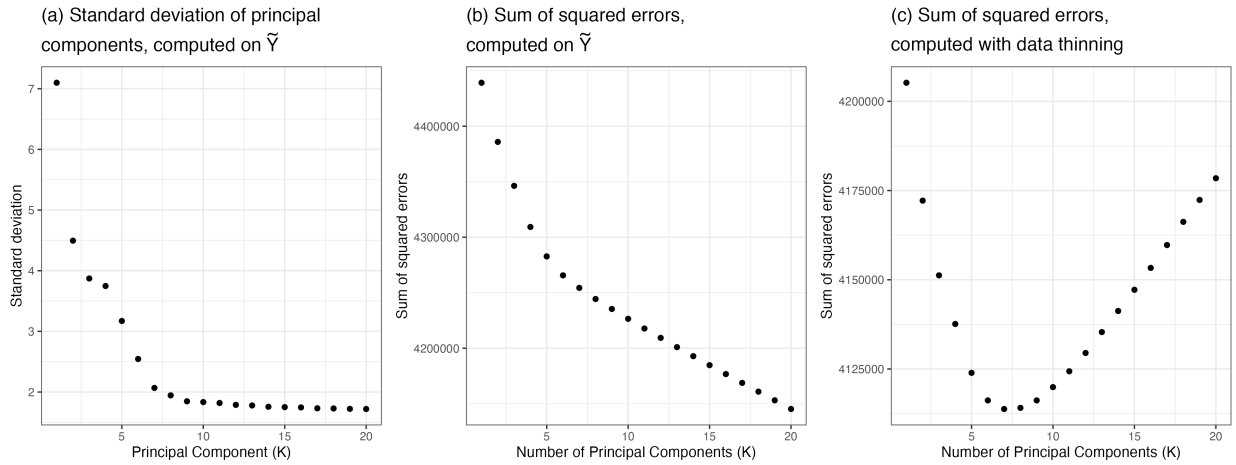


Figure 5.6: Results for the data analysis in Section 5.5. *(a)* An "elbow plot" of the standard deviation of the principal components, which reproduces the plot given in the Seurat guided clustering tutorial. *(b)* Due to the relationship between the sum of squared errors and the standard deviation of the principal components (see Section D.6), looking for an elbow in (a) is equivalent to looking for an elbow in (b). *(c)* The data-thinning version of (b), which shows a clear minimum in the loss function at 7 principal components.

is evidence that a negative binomial model may be preferable in some settings. It is possible to modify the analysis in this section using negative binomial data thinning, as in Neufeld et al. [2023b] (in preparation).

5.6 Discussion

While we focused on applying data thinning to develop a version of cross-validation that is suitable for unsupervised learning, data thinning can also be applied in supervised settings, and may still provide advantages over sample splitting in these settings where sample splitting is an option. For example, cross-validation via data thinning might be preferable to cross-validation via sample splitting in supervised settings if the sample size is small and we wish to avoid excluding high-leverage points from the training set [Leiner et al., 2022]. It may also have power advantages to approaches such as sample splitting or selective inference [Fithian et al., 2014] in problems such as inference after variable selection. Finally, it can be used

to estimate a model's test set error for a range of distributions, along the lines of existing work for the normal distribution [Oliveira et al., 2021] and the Poisson distribution [Oliveira et al., 2022], which may be particularly useful in "fixed-X" regression settings.

While we focused on distributions for which the conditional distribution $G_{\lambda_1,\lambda_2,x}$ takes a simple form (see Table 5.2), the framework can easily be used to study convolution-closed distributions for which $G_{\lambda_1,\lambda_2,x}$ takes an unfamiliar form; we just need to sample from it.

In Section 5.2.4, we considered the impact of using the incorrect value of a nuisance parameter when performing data thinning, but we did not consider what happens when the nuisance parameter is estimated using the data itself. In future work, we will consider the theoretical and empirical implications of performing data thinning with an estimated nuisance parameter. Furthermore, we focused on convolution-closed distributions and thus used additive decompositions where $X = X^{(1)} + X^{(2)}$. For distributions with bounded support, such as the Beta distribution, non-additive decompositions are needed. We leave such decompositions to future work.

An R package implementing data thinning and scripts to reproduce the results in this chapter are available at https://anna-neufeld.github.io/datathin/.

Chapter 6

DISCUSSION

Throughout this dissertation, we saw that *double dipping* is a common pitfall in data analysis that can lead to inflated Type 1 error rates or lead us to select models that overfit the observed data. In this dissertation, we considered three general strategies for avoiding double dipping; selective inference, sample splitting, and data thinning. In this section, we review the advantages and disadvantages of the three approaches, and then describe some ongoing work that addresses the disadvantages of data thinning. This discussion includes material from Dharamshi et al. [2023].

6.1 Review of approaches for addressing double dipping

In Chapter 2, we saw that *selective inference* can provide an elegant way to test data-generated hypotheses while directly accounting for specific, pre-specified hypothesis generating mechanisms. The key advantage of such tailored approaches is that no information is lost at the hypothesis selection stage. In the context of Chapter 2, this meant that an analyst could fit a CART tree to their entire dataset and still subsequently conduct inference on the given CART tree. This may be satisfying to a data analyst, who wishes to test a hypothesis about a given tree that they have already fit to their entire dataset, not an inferior tree built to only half of the data that may suggest entirely different hypotheses. However, selective inference also has numerous disadvantages. The price of using all of our data for hypothesis generation is that our power to test these hypotheses can be quite low in practice (see Section B.3). More importantly, selective inference approaches are very inflexible; an analyst must be carrying out a very rigid (and likely very simple) hypothesis generation strategy, and it must be one that already has a selective inference paper written

about it. This hypothesis selection procedure must be pre-specified, and must be amendable to analytical characterization. Finally, selective inference approaches (including the one in Chapter 2) often require that the data follow a Gaussian distribution.

We briefly note that, while not the focus of this dissertation, there are also highly specific and elegant ways to account for double dipping in the context of model evaluation and selection. These share many of the same advantages and drawbacks as selective inference. For examples, proposals to de-bias the "in-sample" estimate of expected prediction error tend to be specialized to simple, pre-specified models, and thus do not provide an all-purpose tool that is broadly applicable to complex contemporary settings [Oliveira et al., 2021].

On the other end of the extreme, sample splitting is a very flexible tool that is a staple of modern statistics and machine learning courses. Unlike selective inference or specific approaches to de-biasing in-sample prediction error, sample splitting can be used to avoid double dipping in a wide variety of settings. This includes settings where the hypothesis selection procedure cannot be described a priori; i.e. it can involve an expert looking at the data. Sample splitting can also be used in non-parametric settings with no distributional assumptions.

One downside of sample splitting compared to selective inference is that we only get to use a portion of the data for hypothesis selection. As different splits of the data can produce different hypotheses, two analysts studying the same data could end up answering different questions. As pointed out by Rasines and Young [2022], this is not a good reason to discount sample splitting entirely. Whenever our data are randomly sampled from a population, hypothesis selection based on the full data is subject to the same type of issue– a new analyst analyzing a new dataset could end up answering a different question. However, sample splitting still suffers from some additional drawbacks.

1. If the data contain outliers, then each outlier is assigned to a single subsample.

2. If one is interested in drawing conclusions at a per-observation level, then sample splitting is unsuitable. For example, if sample splitting is applied to a dataset consisting

of the 50 states of the United States, then one can only conduct inference or perform validation on those states not used in fitting.

3. If the model of interest is fit using unsupervised learning, then sample splitting may not be applicable. This is discussed in Chapters 3, 4, and 5 of this thesis, as well as in Gao et al. [2022], Chen and Witten [2023], Fu and Perry [2020] and Owen and Perry [2009].

4. In a fixed-covariate regression setting, the interpretation of the coefficients changes between the training and test set, leading to challenges with interpretation.

In Chapters 3, 4, and 5, we developed *count splitting* and then generalized the idea to *data thinning*. Broadly, data thinning shares many advantages with sample splitting; it is a flexible wrapper that can be used to avoid double dipping in a wide variety of settings. Furthermore, by splitting a dataset with n observations into a training set and a test that each have entries corresponding to the same n observations, it avoids many of the drawbacks of sample splitting mentioned above. For example, as emphasized in Chapters 3, 4, and 5, data thinning can be used in unsupervised settings. Furthermore, while not the focus of this thesis, data thinning can be particularly useful in fixed-X regression settings, or in cases where we wish to avoid assigning high-leverage points or outliers to only the training set or only the test set.

Compared to sample splitting, a major disadvantage of data thinning, as described in Chapter 5, is that it is limited to the class of convolution-closed distributions. Furthermore, carrying out the thinning itself often requires knowledge of (potentially unknown) parameters. Working on addressing these drawbacks is an active area of ongoing work. In particular, in Dharamshi et al. [2023], we expand the set of distributions that can be thinned and formally characterize conditions that must be met if we wish to be able to thin without knowledge of the parameters of interest. In the next section, we briefly describe the contents of this ongoing work.

6.2 Ongoing work on generalized data thinning

In Chapter 5, we proposed thinning a random variable X drawn from a convolution-closed family into M independent random variables $X^{(1)}, \ldots, X^{(M)}$ such that $X = \sum_{m=1}^{M} X^{(m)}$ and $X^{(1)}, \ldots, X^{(M)}$ come from the same family of distributions as X. The property that $X = \sum_{m=1}^{M} X^{(M)}$ is desirable because it ensures that no information has been lost in the thinning process. However, this would be equally true if we were to replace the summation by any other deterministic function $T(\cdot)$. Likewise, the fact that $X^{(1)}, \ldots, X^{(M)}$ are from the same family as X, while convenient, is nonessential. Our generalized thinning proposal thus seeks to split X into M random variables such that the following two properties hold:

$$\text{(i) } X = T(X^{(1)}, \ldots, X^{(M)}); \text{ and (ii) } X^{(1)}, \ldots, X^{(M)} \text{ are mutually independent.}$$

It turns out that this generalization is broad enough to simultaneously encompass both convolution-closed data thinning (if X is drawn from a convolution-closed distribution and the function $T(\cdot)$ is addition) and sample splitting (if X is a vector containing multiple independent observation and the function $T(\cdot)$ is concatenation). The added flexibility of this definition greatly increases the scope of distributions that can be thinned. For example, generalized thinning enables us to thin $X \sim \text{Unif}(0, \theta)$ into $X^{(m)} \overset{\text{iid}}{\sim} \theta \cdot \text{Beta}\left(\frac{1}{M}, 1\right)$, for $m = 1, \ldots, M$, in such a way that $X = \max\{X^{(1)}, \ldots, X^{(m)}\}$. A summary of distributions covered by this work is provided in Table 6.1.

We use the following recipe for generalized data thinning. Suppose we know that if $X_m \overset{\text{ind.}}{\sim} Q_\theta^{(1)}$ for $m = 1, \ldots, M$, then $T(X_1, \ldots, X_M) \sim P_\theta$ for some distributions $Q_\theta^{(1)}, \ldots, Q_\theta^{(M)}, P_\theta$ that are all indexed by parameter θ. Then, if we observe realization x of $X \sim P_\theta$, we can obtain independent random variables $X^{(1)}, \ldots, X^{(M)}$ by sampling them from the conditional distribution of $(X_1, \ldots, X_M) \mid T(X_1, \ldots, X_M) = x$. A key contribution of Dharamshi et al. [2023] is characterizing when we can sample from this distribution without knowledge of the parameter θ. It turns out that the key property is *sufficiency*. If $T(X^{(1)}, \ldots, X^{(M)})$ is a

Family	Distribution P_θ, where $X \sim P_\theta$.	Distribution $Q_\theta^{(m)}$ where $X^{(m)} \overset{ind.}{\sim} Q_\theta^{(m)}$.	Sufficient statistic T (sufficient for θ)	Reference / notes
Natural exponential family (in parameter θ)	$N(\theta, \sigma^2)$ Poisson(θ) NegBin(r, θ) Binomial(r, θ) Gamma(α, θ) N$_p(\boldsymbol{\theta}, \Sigma)$ Multinomial$_p(r, \boldsymbol{\theta})$	N$(\epsilon_m\theta, \epsilon_m\sigma^2)$ Poisson$(\epsilon_m\theta)$ NegBin$(\epsilon_m r, \theta)$ Binomial$(\epsilon_m r, \theta)$ Gamma$(\epsilon_m\alpha, \theta)$ $N_p(\epsilon_m\boldsymbol{\theta}, \epsilon_m\Sigma)$ Multinomial$_p(\epsilon_m r, \boldsymbol{\theta})$	$\sum_{m=1}^M X^{(m)}$	Neufeld et al. [2023a]
	Gamma$(K/2, \theta)$	$N(0, \frac{1}{2\theta})$	$\sum_{m=1}^M \left(X^{(m)}\right)^2$	Dharamshi et al. [2023]
	Gamma(K, θ)	Weibull$(\theta^{-\frac{1}{\nu}}, \nu)$	$\sum_{m=1}^M \left(X^{(m)}\right)^\nu$	
General exponential family (in parameter θ)	Beta(θ, β)	Beta$\left(\frac{1}{M}\theta + \frac{m-1}{M}, \frac{1}{M}\beta\right)$	$\left(\Pi_{m=1}^M X^{(m)}\right)^{1/M}$	
	Beta(α, θ)	Beta$\left(\frac{1}{M}\alpha, \frac{1}{M}\theta + \frac{m-1}{M}\right)$	$\left(\Pi_{m=1}^M \left(1 - X^{(m)}\right)\right)^{1/M}$	
	Gamma(θ, β)	Gamma$(\frac{1}{M}\theta + \frac{m-1}{M}, \frac{1}{M}\beta)$	$\left(\Pi_{m=1}^M X^{(m)}\right)^{1/M}$	
	Weibull(θ, ν)	Gamma$(\frac{1}{M}, \theta^{-\nu})$	$\left(\sum_{m=1}^M X^{(m)}\right)^{1/\nu}$	
	Pareto(ν, θ)	Gamma$(\frac{1}{M}, \theta)$	$\nu \times \text{Exp}\left(\sum_{m=1}^M X^{(m)}\right)$	
	N$(0, \theta)$	Gamma$(\frac{1}{2K}, \frac{1}{2\theta})$	$X^2 = \sum_{m=1}^M X^{(m)}$	
	N$_m(\theta_1 1_m, \theta_2 I_m)$	$N(\theta_1, \theta_2)$	sample mean and variance	
Truncated support family	Unif$(0, \theta)$ $\theta \cdot$ Beta$(\alpha, 1)$	$\theta \cdot$ Beta$(\frac{1}{K}, 1)$ $\theta \cdot$ Beta$(\frac{\alpha}{K}, 1)$	$\max\left(X^{(1)}, \ldots, X^{(m)}\right)$	
	$\theta + \text{Exp}(\lambda)$	$\theta + \text{Exp}(\lambda/\text{K})$	$\min\left(X^{(1)}, \ldots, X^{(m)}\right)$	
Non-parametric	F^n	F^{n_m}	The sorted and concatenated elements.	

Table 6.1: Examples of named families (indexed by an unknown parameter θ) that can be thinned into M components without knowledge of the parameter θ, where M is a positive integer. In cases where they are used, $\epsilon_m > 0$ is a tuning parameter such that $\sum_{m=1}^M \epsilon_m = 1$, $n_1, \ldots, n_m$ are integers that sum to n, and $\nu > 0$.

sufficient statistic for θ on the basis of $(X^{(1)}, \ldots, X^{(M)})$, then the conditional distribution we need to sample from will not depend on θ.

This crucial observation sheds important light on the results from Chapter 5 of this thesis, and lets us describe the subset of convolution-closed families that can be thinned without knowledge of the parameters of interest. Recall that the Cauchy family, Cauchy(θ_1, θ_2), indexed by $\theta = (\theta_1, \theta_2)$, is convolution-closed. In particular, if $X^{(1)}, X^{(2)} \overset{iid}{\sim} \text{Cauchy}\left(\frac{1}{2}\theta_1, \frac{1}{2}\theta_2\right)$, then $X^{(1)} + X^{(2)} \sim \text{Cauchy}(\theta_1, \theta_2)$. However, the sum $X^{(1)} + X^{(2)}$ is not sufficient for either θ_1 or θ_2 on the basis of $X^{(1)}$ and $X^{(2)}$. Thus, the main result from Dharamshi et al. [2023] says that we cannot thin a single Cauchy random variable into two independent Cauchy random variables without knowledge of both unknown parameters.

Jørgensen and Song [1998] point out that all convolution-closed distributions that have moment generating functions can be written as (additive) exponential dispersion families $F_{\theta,\lambda}$, indexed by a canonical parameter θ and a dispersion parameter $\lambda > 0$. Distributions in these two parameter families have densities of the form

$$f_X(x \mid \theta, \lambda) = h^*(\lambda, x) e^{\theta x - \lambda A(\theta)}. \tag{6.1}$$

In these families, if $X^{(m)} \overset{\text{ind.}}{\sim} F_{\theta,\lambda_m}$ for $m = 1, \ldots, M$, then $X = \sum_{m=1}^{M} X^{(m)} \sim F_{\theta, \sum_{m=1}^{M} \lambda_m}$. Thus, these families are always convolution-closed in the parameter λ if θ is fixed. Furthermore, $\sum_{m=1}^{M} X^{(m)}$ is always sufficient for θ in the joint distribution of $X^{(1)}, \ldots, X^{(M)}$ [Jørgensen and Song, 1998]. This tells us that, when the distributions from Chapter 5 are written in their additive exponential dispersion family notation, we can always carry out data thinning without knowledge of the canonical parameter θ, but we likely need to know the value of the dispersion parameter λ. We note that an exponential dispersion family with known dispersion parameter λ is a natural exponential family. Thus, the recipe from Chapter 5 essentially provides a recipe for thinning natural exponential families using addition.

We note that, as sufficient statistics are not unique, natural exponential families can also be thinned by alternate, non-additive functions $T(\cdot)$. We also note that not all natural ex-

ponential families can be thinned. As explained in Dharamshi et al. [2023], the Bernoulli(θ) distribution cannot be thinned into independent pieces that each contain non-zero information about θ without knowledge of θ.

Generalized data thinning proposal encompasses a diverse set of existing approaches for splitting a random variable into independent random variables, from convolution-closed data thinning [Neufeld et al., 2023a] to sample splitting [Cox, 1975]. It provides a lens through which these existing approaches follow from the same simple principle — sufficiency — and can be derived through the same simple recipe. The principle of sufficiency is key to generalized data thinning, as it enables a sampling mechanism that does not depend on unknown parameters. When no sufficient statistic that reduces the data is available, as in the Cauchy example above, then sample splitting is still possible. Conversely, in a setting with $n = 1$ or where the elements of $X = (X_1, \ldots, X_n)$ are not independent and identically distributed, sample splitting may not be possible, but other generalized thinning approaches may be available.

6.3 Future work

Despite the recent progress of generalized data thinning, there are still many unanswered questions to be addressed in future work. For example, the starting place for any generalized thinning strategy is the modeling assumption that the data are drawn from a distribution belonging to a family $\mathcal{P}$, which allows us to specify the thinning function $T(\cdot)$ and sampling strategy. We provided results about applying data thinning with the "wrong" values of the nuisance parameters in Chapter 5, but we did not study the effect of estimating nuisance parameters from the data itself. Furthermore, we did not consider alternative types of model misspecification, and their effects on data thinning. Rasines and Young [2022] provide conditions under which their version of Gaussian data thinning will lead to asymptotically valid inference after model selection if the data are non-Gaussian. We hope to provide similar results for other types of data thinning and other types of problems in future work.

BIBLIOGRAPHY

Nadim Aizarani, Antonio Saviano, Laurent Mailly, Sarah Durand, Josip S Herman, Patrick Pessaux, Thomas F Baumert, Dominic Grün, et al. A human liver cell atlas reveals heterogeneity and epithelial progenitors. *Nature*, 572(7768):199–204, 2019.

Susan Athey and Guido Imbens. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27):7353–7360, 2016.

Joshua Batson, Loïc Royer, and James Webber. Molecular cross-validation for single-cell RNA-seq. *BioRxiv*, page 786269, 2019.

PK Bhattacharya. Some aspects of change-point analysis. *Lecture Notes-Monograph Series*, pages 28–56, 1994.

Richard Bourgon. *Overview of the intervals package*, 2009. R Vignette, URL `https://cran.r-project.org/web/packages/intervals/vignettes/intervals_overview.pdf`.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC Press, 1984.

Rasmus Bro, Karin Kjeldahl, Age K Smilde, and HAL Kiers. Cross-validation of component models: a critical look at current methods. *Analytical and Bioanalytical Chemistry*, 390 (5):1241–1251, 2008.

Junyue Cao, Diana R ODay, Hannah A Pliner, Paul D Kingsley, Mei Deng, Riza M Daza, Michael A Zager, Kimberly A Aldinger, Ronnie Blecher-Gonen, Fan Zhang, et al. A human cell atlas of fetal gene expression. *Science*, 370(6518):eaba7721, 2020.

Fan Chen, Sebastien Roch, Karl Rohe, and Shuqi Yu. Estimating graph dimension with cross-validated eigenvalues. *arXiv preprint arXiv:2108.03336*, 2021.

Shuxiao Chen and Jacob Bien. Valid inference corrected for outlier removal. *Journal of Computational and Graphical Statistics*, 29(2):323–334, 2020.

Yiqun T Chen and Daniela M Witten. Selective inference for k-means clustering. *Journal of Machine Learning Research*, 2023.

Saket Choudhary and Rahul Satija. Comparison and evaluation of statistical error models for scRNA-seq. *Genome Biology*, 23(1):1–20, 2022.

Neo Christopher Chung. Statistical significance of cluster membership for unsupervised evaluation of cell identities. *Bioinformatics*, 36(10):3107–3114, 2020.

Neo Christopher Chung and John D Storey. Statistical significance of variables driving systematic variation in high-dimensional data. *Bioinformatics*, 31(4):545–554, 2015.

Neo Christopher Chung, John D. Storey, and Wei Hao. *Jackstraw: statistical inference for unsupervised learning*, 2021. R package version 1.3.1.

David R Cox. A note on data-splitting for the evaluation of significance levels. *Biometrika*, 62(2):441–444, 1975.

Louise Deconinck, Robrecht Cannoodt, Wouter Saelens, Bart Deplancke, and Yvan Saeys. Recent advances in trajectory inference from single-cell omics data. *Current Opinion in Systems Biology*, 2021.

Ameer Dharamshi, Anna Neufeld, Keshav Motwani, Lucy L Gao, Daniela Witten, and Jacob Bien. Generalized data thinning using sufficient statistics. *arXiv preprint arXiv:2303.12931*, 2023.

Tianyang Liu Dongyuan Song and Huy Nguyen. *PseudotimeDE*, 2021. R package version 1.0.0.

Rick Durrett. *Probability: Theory and Examples*, volume 49. Cambridge University Press, 2019.

Reem Elorbany, Joshua M Popp, Katherine Rhodes, Benjamin J Strober, Kenneth Barr, Guanghao Qi, Yoav Gilad, and Alexis Battle. Single-cell sequencing reveals lineage-specific dynamic genetic regulation of gene expression during human cardiomyocyte differentiation. *PLoS Genetics*, 18(1):e1009666, 2022.

Gökcen Eraslan, Lukas M Simon, Maria Mircea, Nikola S Mueller, and Fabian J Theis. Single-cell RNA-seq denoising using a deep count autoencoder. *Nature Communications*, 10(1):1–14, 2019.

William Fithian, Dennis Sun, and Jonathan Taylor. Optimal inference after model selection. *arXiv preprint arXiv:1410.2597*, 2014.

Wei Fu and Patrick O Perry. Estimating the number of clusters using cross-validation. *Journal of Computational and Graphical Statistics*, 29(1):162–173, 2020.

Lucy L Gao, Jacob Bien, and Daniela Witten. Selective inference for hierarchical clustering. *Journal of the American Statistical Association*, pages 1–11, 2022.

David Gerard. Data-based RNA-seq simulations by binomial thinning. *BMC Bioinformatics*, 21(1):1–14, 2020.

Isabella N Grabski, Kelly Street, and Rafael A Irizarry. Significance Analysis for Clustering with Single-Cell RNA-Sequencing Data. *bioRxiv*, 2022.

Dominic Grün, Anna Lyubimova, Lennart Kester, Kay Wiebrands, Onur Basak, Nobuo Sasaki, Hans Clevers, and Alexander Van Oudenaarden. Single-cell messenger RNA sequencing reveals rare intestinal cell types. *Nature*, 525(7568):251–255, 2015.

Christoph Hafemeister and Rahul Satija. Normalization and variance stabilization of single-cell RNA-seq data using regularized negative binomial regression. *Genome Biology*, 20(1): 1–15, 2019.

Laleh Haghverdi, Aaron TL Lun, Michael D Morgan, and John C Marioni. Batch effects in single-cell rna-sequencing data are corrected by matching mutual nearest neighbors. *Nature Biotechnology*, 36(5):421–427, 2018.

Yuhan Hao, Stephanie Hao, Erica Andersen-Nissen, William M. Mauck III, Shiwei Zheng, Andrew Butler, et al. Integrated analysis of multimodal single-cell data. *Cell*, 2021.

Peter Harremoës, Oliver Johnson, and Ioannis Kontoyiannis. Thinning, entropy, and the law of thin numbers. *IEEE Transactions on Information Theory*, 56(9):4228–4244, 2010.

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer, 2009.

Paul Hoffman et al. *Seurat*, 2021. R package version 4.0.6. https://satijalab.org/seurat/index.html.

Paul Hoffman et al. Seurat - guided clustering tutorial. https://satijalab.org/seurat/articles/pbmc3k_tutorial.html, 2022. Accessed: 09-12-2022.

Torsten Hothorn and Achim Zeileis. partykit: A modular toolkit for recursive partytioning in R. *The Journal of Machine Learning Research*, 16(1):3905–3909, 2015.

Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3): 651–674, 2006.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1): 193–218, 1985.

Sangwon Hyun, Max G'Sell, and Ryan J Tibshirani. Exact post-selection inference for changepoint detection and other generalized lasso problems. *arXiv preprint arXiv:1606.03552*, 2016.

Sangwon Hyun, Kevin Z Lin, Max G'Sell, and Ryan J Tibshirani. Post-selection inference for changepoint detection algorithms with application to copy number variation data. *Biometrics*, pages 1–13, 2021.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 112. Springer, 2013.

Sean Jewell, Paul Fearnhead, and Daniela Witten. Testing for a change in mean after changepoint detection. *Journal of the Royal Statistical Society, Series B*, 2022.

Harry Joe. Time series models with univariate margins in the convolution-closed infinitely divisible class. *Journal of Applied Probability*, 33(3):664–677, 1996.

Bent Jørgensen. Exponential dispersion models and extensions: A review. *International Statistical Review/Revue Internationale de Statistique*, pages 5–20, 1992.

Bent Jørgensen and Peter Xue-Kun Song. Stationary time series models with exponential dispersion model margins. *Journal of Applied Probability*, 35(1):78–92, 1998.

George Kimeldorf, Detlef Plachky, and Allan R Sampson. A simultaneous characterization of the Poisson and Bernoulli distributions. *Journal of Applied Probability*, 18(1):316–320, 1981.

Danijel Kivaranovic and Hannes Leeb. On the length of post-model-selection confidence intervals conditional on polyhedral constraints. *Journal of the American Statistical Association*, 116(534):845–857, 2021.

David Lähnemann, Johannes Köster, Ewa Szczurek, Davis J McCarthy, Stephanie C Hicks, Mark D Robinson, Catalina A Vallejos, Kieran R Campbell, Niko Beerenwinkel, Ahmed Mahfouz, et al. Eleven grand challenges in single-cell data science. *Genome Biology*, 21 (1):1–35, 2020.

Tilman Lange, Volker Roth, Mikio L Braun, and Joachim M Buhmann. Stability-based validation of clustering solutions. *Neural Computation*, 16(6):1299–1323, 2004.

Jason D Lee, Dennis L Sun, Yuekai Sun, and Jonathan E Taylor. Exact post-selection inference, with application to the lasso. *The Annals of Statistics*, 44(3):907–927, 2016.

James Leiner, Boyan Duan, Larry Wasserman, and Aaditya Ramdas. Data fission: splitting a single data point. *arXiv preprint arXiv:2112.11079*, 2022.

Keli Liu, Jelena Markovic, and Robert Tibshirani. More powerful post-selection inference, with application to the lasso. *arXiv preprint arXiv:1801.09037*, 2018.

Joshua R Loftus and Jonathan E Taylor. Selective inference in regression models with groups of variables. *arXiv preprint arXiv:1511.01478*, 2015.

Wei-Yin Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348, 2014.

Wei-Yin Loh, Haoda Fu, Michael Man, Victoria Champion, and Menggang Yu. Identification of subgroups with differential treatment effects for longitudinal and multiresponse variables. *Statistics in Medicine*, 35(26):4837–4855, 2016.

Wei-Yin Loh, Michael Man, and Shuaicheng Wang. Subgroups from regression trees with adjustment for prognostic effects and postselection inference. *Statistics in Medicine*, 38 (4):545–557, 2019.

Romain Lopez, Jeffrey Regier, Michael B Cole, Michael I Jordan, and Nir Yosef. Deep

generative modeling for single-cell transcriptomics. *Nature Methods*, 15(12):1053–1058, 2018.

Francisco Louzada, Pedro L. Ramos, and Eduardo Ramos. A note on bias of closed-form estimators for the gamma distribution derived from likelihood equations. *The American Statistician*, 73(2):195–199, 2019.

Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15(12):1–21, 2014.

Aaron T. L. Lun, Davis J. McCarthy, and John C. Marioni. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Res.*, 5:2122, 2016.

Ed McKenzie. Autoregressive moving-average processes with negative-binomial and geometric marginal distributions. *Advances in Applied Probability*, 18(3):679–705, 1986.

Anna Neufeld, Lucy L Gao, Joshua Popp, Alexis Battle, and Daniela Witten. Inference after latent variable estimation for single-cell RNA sequencing data. *Biostatistics*, 2022a.

Anna Neufeld, Ameer Dharamshi, Lucy L Gao, and Daniela Witten. Data thinning for convolution-closed distributions. *arXiv preprint arXiv:2301.07276*, 2023a.

Anna Neufeld, Lucy Gao, Joshua Popp, Alexis Battle, and Daniela Witten. Negative binomial count splitting for single-cell RNA sequencing data. *To be submitted*, 2023b.

Anna C Neufeld, Lucy L Gao, and Daniela M Witten. Tree-values: selective inference for regression trees. *Journal of Machine Learning Research*, 23(305):1–43, 2022b.

Natalia L Oliveira, Jing Lei, and Ryan J Tibshirani. Unbiased risk estimation in the normal means problem via coupled bootstrap techniques. *arXiv preprint arXiv:2111.09447*, 2021.

Natalia L Oliveira, Jing Lei, and Ryan J Tibshirani. Coupled bootstrap test error estimation for Poisson variables. *arXiv preprint arXiv:2212.01943*, 2022.

Art B Owen and Patrick O Perry. Bi-cross-validation of the svd and the nonnegative matrix factorization. *The Annals of Applied Statistics*, 3(2):564–594, 2009.

Hannah Pliner, Xiaojie Qiu, Cole Trapnell, and Brent Ewing. *Monocle3*, 2020. R package version 1.2.9. https://cole-trapnell-lab.github.io/monocle3/.

Hannah Pliner, Xiaojie Qiu, Cole Trapnell, and Others. Monocle3 tutorials: Differential expression analysis. https://cole-trapnell-lab.github.io/monocle3/docs/differential/, 2022. Accessed: 09-12-2022.

D Garca Rasines and G A Young. Splitting strategies for post-selection inference. *Biometrika*, 12 2022. ISSN 1464-3510. doi: 10.1093/biomet/asac070.

Brian D Ripley. *Pattern recognition and neural networks*. Cambridge University Press, 1996.

Abhishek Sarkar and Matthew Stephens. Separating measurement and expression models clarifies confusion in single-cell RNA sequencing analysis. *Nature Genetics*, 53(6):770–777, 2021.

Rahul Satija, Jeffrey A Farrell, David Gennert, Alexander F Schier, and Aviv Regev. Spatial reconstruction of single-cell gene expression data. *Nature Biotechnology*, 33:495–502, 2015.

Mark Robert Segal. Regression trees for censored data. *Biometrics*, 44(1):35–47, 1988.

Dongyuan Song and Jingyi Jessica Li. Pseudotimede: inference of differential gene expression along cell pseudotime with well-calibrated p-values from single-cell rna sequencing data. *Genome Biology*, 22(1):1–25, 2021.

Tim Stuart, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck III, et al. Comprehensive integration of single-cell data. *Cell*, 177: 1888–1902, 2019.

Jonathan Taylor and Robert J Tibshirani. Statistical learning and selective inference. *Proceedings of the National Academy of Sciences*, 112(25):7629–7634, 2015.

Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2019. R package version 4.1-15, available on CRAN.

Xiaoying Tian and Jonathan Taylor. Asymptotics of selective inference. *Scandinavian Journal of Statistics*, 44(2):480–499, 2017.

Xiaoying Tian and Jonathan Taylor. Selective inference with a randomized response. *The Annals of Statistics*, 46(2):679–710, 2018.

Robert Tibshirani and Guenther Walther. Cluster validation by prediction strength. *Journal of Computational and Graphical Statistics*, 14(3):511–528, 2005.

Ryan J Tibshirani, Jonathan Taylor, Richard Lockhart, and Robert Tibshirani. Exact post-selection inference for sequential regression procedures. *Journal of the American Statistical Association*, 111(514):600–620, 2016.

Ryan J Tibshirani, Alessandro Rinaldo, Rob Tibshirani, and Larry Wasserman. Uniform asymptotic inference and the bootstrap after model selection. *The Annals of Statistics*, 46 (3):1255–1287, 2018.

F. W. Townes, S. C. Hicks, M. J. Aryee, and R. A. Irizarry. Feature selection and dimension reduction for single-cell RNA-Seq based on a multinomial model. *Genome Biology*, 20(1): 1–16, 2019.

Cole Trapnell, Davide Cacchiarelli, Jonna Grimsby, Prapti Pokharel, Shuqiang Li, Michael Morse, Niall J Lennon, Kenneth J Livak, Tarjei S Mikkelsen, and John L Rinn. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature Biotechnology*, 32(4):381–386, 2014.

Theresa Ullmann, Christian Hennig, and Anne-Laure Boulesteix. Validation of cluster analysis results on validation data: A systematic framework. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(3):e1444, 2022.

Koen Van den Berge, Hector Roux De Bezieux, Kelly Street, Wouter Saelens, Robrecht Cannoodt, Yvan Saeys, Sandrine Dudoit, and Lieven Clement. Trajectory-based differential expression analysis for single-cell sequencing data. *Nature Communications*, 11(1):1–13, 2020.

Ashwini Venkatasubramaniam and Julian Wolfson. *visTree: Visualization of Subgroups for Decision Trees*, 2018. R package version 0.8.1, available on CRAN.

Ashwini Venkatasubramaniam, Julian Wolfson, Nathan Mitchell, Timothy Barnes, Meghan JaKa, and Simone French. Decision trees in epidemiological research. *Emerging Themes in Epidemiology*, 14(1):11, 2017.

Stefan Wager and Guenther Walther. Adaptive concentration of regression trees, with application to random forests. *arXiv preprint arXiv:1503.06388*, 2015.

Jon Wakefield. *Bayesian and frequentist regression methods*, volume 23. Springer, 2013.

Jingshu Wang, Mo Huang, Eduardo Torre, Hannah Dueck, Sydney Shaffer, John Murray, et al. Gene expression distribution deconvolution in single-cell RNA sequencing. *Proceedings of the National Academy of Sciences*, 115(28):E6437–E6446, 2018.

Zhi-Sheng Ye and Nan Chen. Closed-form estimators for the gamma distribution derived from likelihood equations. *The American Statistician*, 71(2):177–181, 2017.

Jesse M Zhang, Govinda M Kamath, and N Tse David. Valid post-clustering differential analysis for single-cell RNA-Seq. *Cell Systems*, 9(4):383–392, 2019.

Shijie C. Zheng, Genevieve Stein-O'Brien, Jonathan J. Augustin, Jared Slosberg, Giovanni A. Carosso, Briana Winer, Gloria Shin, Hans T. Bjornsson, Loyal A. Goff, and Kasper D. Hansen. Universal prediction of cell cycle position using transfer learning. *Genome Biology*, 23:41, 2022. doi: 10.1186/s13059-021-02581-y.

Appendix A

APPENDICES FOR CHAPTER 2

A.1 Comparison to Loh et al. [2019]

Loh et al. [2016] and Loh et al. [2019] use regression trees to find subgroups of patients with similar treatment effects in clinical trials. They grow trees based on patient characteristics using a different algorithm than CART. Furthermore, they are interested in the mean treatment effect (which is a linear regression coefficient) within each terminal region of the tree, rather than the mean response within each region.

One of the goals of Loh et al. [2019] is to construct valid post-selection confidence intervals for the treatment effect within each terminal node. In this appendix, we show that their approach, when adapted to the setting of this paper, does not yield confidence intervals with nominal coverage.

The basic idea of Loh et al. [2019], instantiated to our setting, is as follows. Suppose that $R_A \in \textsc{Tree}^{\lambda}(y)$, and define the vector ν_{reg} such that $(\nu_{reg})_i = 1_{(x_i \in R_A)} / \left\{\sum_{i'=1}^{n} 1_{(x_{i'} \in R_A)}\right\}$, as in (2.14). We know that the "naive" Z-interval does not achieve nominal coverage, meaning that

$$\Pr\left(\nu_{reg}^{\top}\mu \in \left[\nu_{reg}^{\top}y - z_{\alpha/2}\frac{\sigma}{\sqrt{\sum_{i=1}^{n} 1_{(x_i \in R_A)}}}, \nu_{reg}^{\top}y + z_{\alpha/2}\frac{\sigma}{\sqrt{\sum_{i=1}^{n} 1_{(x_i \in R_A)}}}\right]\right) < 1 - \alpha. \quad \text{(A.1)}$$

This is because the "multiplier" for the naive confidence interval, $z_{\alpha/2}$, is derived under the assumption that the region R_A (or, equivalently, the vector ν_{reg}), is fixed, rather than a function of the data.

Loh et al. [2019] observe that there exists some $\alpha' < \alpha$ such that

$$\Pr\left(\nu_{reg}^{\top}\mu \in \left[\nu_{reg}^{\top}y - z_{\alpha'/2}\frac{\sigma}{\sum_{i=1}^{n} 1_{(x_i \in R_A)}}, \nu_{reg}^{\top}y + z_{\alpha'/2}\frac{\sigma}{\sum_{i=1}^{n} 1_{(x_i \in R_A)}}\right]\right) = 1 - \alpha. \quad \text{(A.2)}$$

The value for α' for a given tree will depend on the number of split covariates p, the number of data points n, the depth of the tree, and the value of λ used for tree pruning, among other considerations. If we know how the data was generated, then we can check whether some value α' satisfies (A.2) as follows:

1. Draw B different simulated datasets $\{(X_b, y_b)\}_{b=1}^{B}$ from the same distribution (call this F) as the original data. For $b = 1, \ldots, B$:

   (a) Build a tree using the simulated data (X_b, y_b), using the same procedure and the same settings as in Step 1, and denote it $\textsc{tree}^{\lambda}(y_b)$.

   (b) For each terminal region $R \in \textsc{term}\left(\textsc{tree}^{\lambda}(y_b), \mathbb{R}^p\right)$ in the tree:

      i. Construct a $(1 - \alpha')$ naive Z-interval for the mean response in the region using (X_b, y_b).

      ii. Check if each interval contains $\bar{\mu}_R$, the true mean for this region R.

2. Compute the fraction of intervals in 1(b) that contain $\bar{\mu}_R$.

3. If this value is $1 - \alpha$, then we have found the correct value of α'. If not, then we try a larger or smaller value of α'.

We test this procedure in a very simple simulation study. We generate data $X_{ij} \sim N(0, 1)$ and $y_i \sim N(0, 1)$ for $i = 1, \ldots, 100$ and $j = 1, \ldots, p$. In this simple setting, the true mean response for every region in every fitted tree is 0. We carry out the procedure outlined above with $\alpha = 0.1$. For two values of p and for CART trees with 1, 2, and 3 levels, we create 1000 datasets and 1000 trees and report the empirical coverage of the intervals obtained using this

ideal method, averaged over all nodes in all trees. The results, shown in Table A.1, show that this ideal procedure procedure leads to intervals that achieve nominal coverage.

Unfortunately, this ideal procedure is practically infeasible, as it requires the user to know the true distribution of the data F. Thus, in practice, Loh et al. [2019] propose replacing F by $\hat{F}$, the empirical distribution of the original data. This amounts to replacing the simulated datasets in Step 2 with bootstrapped datasets, and checking whether the naive Z-intervals in Step 1(b) contain $\bar{y}_R$ rather than $\bar{\mu}_R$.

We can see why this is problematic in a very simple setting where we fit a tree with depth 1. If all observations have mean 0, then a CART tree fit to a bootstrap sample of the data will nevertheless find regions R_L and R_R such that the sample mean value of y_b within R_L is negative and the sample mean value of y_b within R_R is positive. As y_b and y contain many overlapping observations, it is likely that the sample mean value of y within R_L is also negative and the sample mean value of y within R_R is also positive. In other words, because of the overlap between y and y_b, the within-region sample means of y_b are closer to the within-region sample means of y than they are to the within-region population means. Thus, when we calibrate α' to cover the mean values of y within various regions, we end up with under-coverage of the true population mean, as shown in Table A.1.

We see in Table A.1 that our selective inference framework approach enables valid inference in this setting, whereas a bootstrap procedure modeled after Loh et al. [2019] does not.

p	Tree depth	Loh (ideal) Coverage	Loh (ideal) Average α'	Loh (bootstrap) Coverage	Loh (bootstrap) Average α'	Selective CIs Coverage
2	1	0.902	0.008	0.749	0.037	0.890
	2	0.905	0.004	0.695	0.038	0.904
	3	0.900	0.005	0.660	0.047	0.895
20	1	0.883	0.001	0.601	0.016	0.908
	2	0.900	0.00025	0.549	0.016	0.901
	3	0.904	0.00015	0.543	0.022	0.905

Table A.1: Coverage of 90% confidence intervals computed using three methods for the simple setting where $y_i \sim N(0,1)$ and $X_{ij} \sim N(0,1)$ for for $i = 1, \ldots, 100$ and $j = 1, \ldots, p$. Note that the "Loh (ideal)" method can never be used in practice, as it requires knowledge of the true parameter.

A.2 Proofs for Section 2.3

A.2.1 Proof of Theorem 2.1

Let $0 \leq \alpha \leq 1$. We start by proving the first statement in Theorem 2.1:

$$pr_{H_0}\left\{p_{sib}(Y) \leq \alpha \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\right\} = \alpha.$$

This is a special case of Proposition 3 from Fithian et al. [2014]. It follows from the definition of $p_{sib}(Y)$ in (2.8) that

$$pr_{H_0}\left\{p_{sib}(Y) \leq \alpha \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y), \mathcal{P}^{\perp}_{\nu_{sib}} Y = \mathcal{P}^{\perp}_{\nu_{sib}} y\right\} = \alpha.$$

Therefore, applying the law of total expectation yields

$$
\begin{aligned}
&pr_{H_0}\left\{p_{sib}(Y) \leq \alpha \mid R_A, R_B \text{ siblings in } \textsc{tree}^{\lambda}(Y)\right\} \\
&= E_{H_0}\left[1_{\{p_{sib}(Y)<\alpha\}} \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\right] \\
&= E_{H_0}\Bigg(E_{H_0}\left[1_{\{p_{sib}(Y)<\alpha\}} \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y), \mathcal{P}^{\perp}_{\nu_{sib}} Y = \mathcal{P}^{\perp}_{\nu_{sib}} y\right] \\
&\qquad\qquad \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\Bigg) \\
&= E_{H_0}\left\{\alpha \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\right\} = \alpha.
\end{aligned}
$$

The second statement of Theorem 2.1 follows directly from the following result.

Lemma A.1 *If $Y \sim N_n(\mu, \sigma^2 I_n)$, then the random variable $\nu_{sib}^{\mathrm{T}} Y$ has the following conditional distribution:*

$$
\nu_{sib}^{\mathrm{T}} Y \mid \{R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y), \mathcal{P}^{\perp}_{\nu_{sib}} Y = \mathcal{P}^{\perp}_{\nu_{sib}} y\} \sim \mathcal{TN}\left\{\nu_{sib}^{\mathrm{T}}\mu, \sigma^2\|\nu_{sib}\|_2^2; S^{\lambda}_{sib}(\nu_{sib})\right\}, \quad \text{(A.3)}
$$

where $S^{\lambda}_{sib}(\nu_{sib})$ is defined in (2.10) *and $\mathcal{TN}(\mu, \sigma, S)$ denotes the $N(\mu, \sigma^2)$ distribution truncated to the set S.*

A.2.1.0.1 Proof: The following holds for any $\nu \in \mathbb{R}^n$.

$$
\begin{aligned}
&pr\left\{\nu^{\mathrm{T}} Y > c \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y), \mathcal{P}^{\perp}_{\nu} Y = \mathcal{P}^{\perp}_{\nu} y\right\} \\
&= pr\left\{\nu^{\mathrm{T}} Y > c \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}\left(\mathcal{P}^{\perp}_{\nu} Y + \frac{\nu\nu^{\mathrm{T}}}{\|\nu\|_2^2} Y\right), \mathcal{P}^{\perp}_{\nu} Y = \mathcal{P}^{\perp}_{\nu} y\right\} \\
&= pr\left\{\nu^{\mathrm{T}} Y > c \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}\left(\mathcal{P}^{\perp}_{\nu} y + \frac{\nu}{\|\nu\|_2^2}\nu^{\mathrm{T}} Y\right), \mathcal{P}^{\perp}_{\nu} Y = \mathcal{P}^{\perp}_{\nu} y\right\} \\
&= pr\left\{\nu^{\mathrm{T}} Y > c \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}\left(\mathcal{P}^{\perp}_{\nu} y + \frac{\nu}{\|\nu\|_2^2}\nu^{\mathrm{T}} Y\right)\right\} \\
&= pr\left\{\phi > c \mid \phi \in S^{\lambda}_{sib}(\nu)\right\},
\end{aligned}
$$

where $\phi = \nu^{\mathrm{T}} Y$. In the fourth line, the condition $\mathcal{P}^{\perp}_{\nu} Y = \mathcal{P}^{\perp}_{\nu} y$ can be dropped because when $Y \sim N_n(\mu, \sigma^2 I_n)$, $\mathcal{P}^{\perp}_{\nu} Y$ is independent of $\nu^{\top} Y$. Finally, since $\phi \sim N(\nu^{\mathrm{T}}\mu, \sigma^2\|\nu\|_2^2)$, (A.3)

holds. □

A.2.2 Proof of Proposition 2.3.1

Theorem 6.1 from Lee et al. [2016] says that the truncated normal distribution has monotone likelihood ratio in the mean parameter. This guarantees that $L(y)$ and $U(y)$ in (2.13) are unique. Then, for $L(\cdot)$ and $U(\cdot)$ in (2.13), (A.3) in Lemma A.1 guarantees that

$$pr\left\{\nu^{\mathrm{T}}\mu \in [L(Y), U(Y)] \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y), \mathcal{P}_{\nu}^{\perp} Y = \mathcal{P}_{\nu}^{\perp} y\right\} = 1 - \alpha. \tag{A.4}$$

Finally, we need to prove that (A.4) implies $(1-\alpha)$–*selective coverage* as defined in (2.12). Following Proposition 3 from Fithian et al. [2014], let η be the random variable $\mathcal{P}_{\nu}^{\perp} Y$ and let $f(\cdot)$ be its density. Then,

$$\begin{aligned}
&pr\left\{\nu^{\mathrm{T}}\mu \in [L(Y), U(Y)] \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y)\right\} \\
&= \int pr\left\{\nu^{\mathrm{T}}\mu \in [L(Y), U(Y)] \mid R_A, R_B \text{ are siblings in } \textsc{tree}^{\lambda}(Y), \mathcal{P}_{\nu}^{\perp} Y = \mathcal{P}_{\nu}^{\perp} y\right\} f(\eta) d\eta \\
&= \int (1-\alpha) f(\eta) d\eta = 1 - \alpha.
\end{aligned}$$

A.2.3 Proof of Theorem 2.2

We omit the proof of the first statement of Theorem 2.2, as it is similar to the proof of the first statement of Theorem 2.1 in Appendix A.2.1.

The second statement in Theorem 2.2 follows directly from the following result.

Lemma A.2 *The random variable $\nu_{reg}^{\mathrm{T}} Y$ has the conditional distribution*

$$\nu_{reg}^{\mathrm{T}} Y \mid \{R_A \in \textsc{tree}^{\lambda}(Y), \mathcal{P}_{\nu_{reg}}^{\perp} Y = \mathcal{P}_{\nu_{reg}}^{\perp} y\} \sim \mathcal{TN}\left\{\nu_{reg}^{\mathrm{T}}\mu, \sigma^2 \|\nu_{reg}\|_2^2; S_{reg}^{\lambda}(\nu_{reg})\right\}, \tag{A.5}$$

where $S_{reg}^{\lambda}(\nu_{reg})$ was defined in (2.16).

We omit the proof of Lemma A.2, as it is similar to the proof of Lemma A.1.

A.2.4 Proof of Proposition 2.3.2

The proof largely follows the proof of Proposition 2.3.1. The fact that the truncated normal distribution has monotone likelihood ratio (Theorem 6.1 of Lee et al. 2016) ensures that $L(y)$ and $U(y)$ defined in (2.17) are unique, and (A.5) in Lemma A.2 implies that

$$pr\left\{\nu_{reg}^{\mathrm{T}}\mu \in [L(Y), U(Y)] \mid R_A \in \mathrm{TREE}^{\lambda}(Y), \mathcal{P}_{\nu_{reg}}^{\perp} Y = \mathcal{P}_{\nu_{reg}}^{\perp} y\right\} = 1 - \alpha.$$

The rest of the argument is as in the proof of Proposition 2.3.1.

A.3 Proofs for Section 2.4.1

A.3.1 Proof of Lemma 2.1

We first state and prove the following lemma.

Lemma A.3 *Let R_A and R_B be the regions in the definition of ν_{sib} in* (2.6)*. For an arbitrary region R and for any $j \in \{1, \ldots, p\}$ and $s \in \{1, \ldots, n-1\}$, recall that the potential children of R (the ones that CART will consider adding to the tree when applying Algorithm 2.1 to region R) are given by $R \cap \chi_{j,s,0}$ and $R \cap \chi_{j,s,1}$, where $\chi_{j,s,0}$ and $\chi_{j,s,l}$ were defined in* (2.1)*. If $(R_A \cup R_B) \subseteq R \cap \chi_{j,s,0}$ or $(R_A \cup R_B) \subseteq R \cap \chi_{j,s,1}$, then $Gain_R\{y'(\phi, \nu_{sib}), j, s\} = Gain_R\{y, j, s\}$ for all ϕ.*

A.3.1.0.1 Proof: It follows from algebra that for $Gain_R(y, j, s)$ defined in (2.2),

$$Gain_R(y,j,s) = -\left\{\sum_{i=1}^{n} 1_{(x_i \in R)}\right\} (\overline{y}_R)^2 + \left\{\sum_{i=1}^{n} 1_{(x_i \in R \cap \chi_{j,s,0})}\right\} \left(\overline{y}_{R \cap \chi_{j,s,0}}\right)^2 + \left\{\sum_{i=1}^{n} 1_{(x_i \in R \cap \chi_{j,s,1})}\right\} \left(\overline{y}_{R \cap \chi_{j,s,1}}\right)^2, \quad \text{(A.6)}$$

where $\bar{y}_R = \left(\sum_{i \in R} y_i\right) / \left\{\sum_{i=1}^{n} 1_{(x_i \in R)}\right\}$. It follows from (A.6) that to prove Lemma A.3, it suffices to show that $\overline{y}_T = \overline{y'(\phi, \nu_{sib})}_T$ for $T \in \{R, R \cap \chi_{j,s,0}, R \cap \chi_{j,s,1}\}$. Recall from

Section 2.3.3 that $\{y'(\phi, \nu_{sib})\}_i = y_i + \Delta_i$, where

$$\Delta_i = \begin{cases} (\phi - \nu_{sib}^{\mathrm{T}} y) \frac{\sum_{i'=1}^n 1_{(x'_i \in R_B)}}{\sum_{i'=1}^n 1_{(x'_i \in R_A \cup R_B)}} & \text{if } i \in R_A \\ -(\phi - \nu_{sib}^{\mathrm{T}} y) \frac{\sum_{i'=1}^n 1_{(x'_i \in R_A)}}{\sum_{i'=1}^n 1_{(x'_i \in R_A \cup R_B)}} & \text{if } i \in R_B \\ 0 & \text{otherwise.} \end{cases}$$

Without loss of generality, assume that $(R_A \cup R_B) \subseteq R \cap \chi_{j,s,0}$. For any $T \in \{R, R \cap \chi_{j,s,0}\}$, $R_A \cup R_B \subseteq T$. Thus,

$$\begin{aligned} \overline{y'(\phi, \nu_{sib})}_T &= \frac{1}{\sum_{i=1}^n 1_{(x_i \in T)}} \left\{ \sum_{i \in T \setminus (R_A \cup R_B)} y_i + \sum_{i \in R_A} (y_i + \Delta_i) + \sum_{i \in R_B} (y_i + \Delta_i) \right\} \\ &= \bar{y}_T + \frac{\sum_{i \in R_A} \Delta_i + \sum_{i \in R_B} \Delta_i}{\sum_{i=1}^n 1_{(x_i \in T)}} \\ &= \bar{y}_T + \frac{\left\{\sum_{i=1}^n 1_{(x_i \in R_A)}\right\} (\phi - \nu_{sib}^{\mathrm{T}} y) \frac{\sum_{i=1}^n 1_{(x_i \in R_B)}}{\sum_{i=1}^n 1_{(x_i \in R_A \cup R_B)}} - \left\{\sum_{i=1}^n 1_{(x_i \in R_B)}\right\} (\phi - \nu_{sib}^{\mathrm{T}} y) \frac{\sum_{i=1}^n 1_{(x_i \in R_A)}}{\sum_{i=1}^n 1_{(x_i \in R_A \cup R_B)}}}{\sum_{i=1}^n 1_{(x_i \in T)}} \\ &= \bar{y}_T + 0 = \bar{y}_T. \end{aligned}$$

Furthermore,

$$\overline{y'(\phi, \nu_{sib})}_{R \cap \chi_{j,s,1}} = \frac{1}{\sum_{i=1}^n 1_{(x_i \in R \cap \chi_{j,s,1})}} \sum_{i \in R \cap \chi_{j,s,1}} (y_i + \Delta_i) = \frac{1}{\sum_{i=1}^n 1_{(x_i \in R \cap \chi_{j,s,1})}} \sum_{i \in R \cap \chi_{j,s,1}} (y_i + 0) = \overline{y}_{R \cap \chi_{j,s,1}}.$$

□

We will now prove Lemma 2.1.

It follows from Definition 2.6 that if $\mathcal{R}\left[\textsc{branch}\{R, \textsc{tree}^\lambda(y)\}\right] \subseteq \textsc{tree}^\lambda\{y'(\phi, \nu_{sib})\}$, then R_A and R_B are siblings in $\textsc{tree}^\lambda\{y'(\phi, \nu_{sib})\}$. This establishes the ($\Leftarrow$) direction.

We will prove the ($\Rightarrow$) direction by contradiction. Suppose that R_A and R_B are siblings in $\textsc{tree}^\lambda\{y'(\phi, \nu_{sib})\}$. Define $\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\} = ((j_1, s_1, e_1), \ldots, (j_L, s_L, e_L))$, and define $R^{(l')} = \bigcap_{l=1}^{l'} \chi_{j_l, s_l, e_l}$ for $l' = 1, \ldots, L$. Assume that there exists $l \in \{0, \ldots, L-2\}$ such that $R^{(l)} \in \textsc{tree}^\lambda\{y'(\phi, \nu_{sib})\}$ and $R^{(l+1)} \notin \textsc{tree}^\lambda\{y'(\phi, \nu_{sib})\}$. We assume that any

ties between splits that occur at Step 2 of Algorithm 2.1 are broken in the same way for y and $y'(\phi, \nu_{sib})$, and so this implies that there exists $(\tilde{j}, \tilde{s}) \neq (j_{l+1}, s_{l+1})$ such that $(\tilde{j}, \tilde{s}) \in \arg\max_{j,s} Gain_{R^{(l)}}\{y'(\phi, \nu_{sib}), j, s\}$ and

$$Gain_{R^{(l)}}\{y'(\phi, \nu_{sib}), j_{l+1}, s_{l+1}\} < Gain_{R^{(l)}}\{y'(\phi, \nu_{sib}), \tilde{j}, \tilde{s}\}. \tag{A.7}$$

Since R_A and R_B are siblings in $\textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}$, it follows from Lemma A.3 that $Gain_{R^{(l)}}\{y'(\phi, \nu_{sib}), \tilde{j}, \tilde{s}\} = Gain_{R^{(l)}}(y, \tilde{j}, \tilde{s})$. Also, since R_A and R_B are siblings in $\textsc{tree}^{\lambda}(y)$, it follows from Lemma A.3 that $Gain_{R^{(l)}}\{y'(\phi, \nu_{sib}), j_{l+1}, s_{l+1}\} = Gain_{R^{(l)}}(y, j_{l+1}, s_{l+1})$. Applying these facts to (A.7) yields

$$Gain_{R^{(l)}}(y, j_{l+1}, s_{l+1}) < Gain_{R^{(l)}}(y, \tilde{j}, \tilde{s}). \tag{A.8}$$

But since $R^{(l)}$ and $R^{(l+1)}$ both appeared in $\textsc{tree}^{\lambda}(y)$,

$$(j_{l+1}, s_{l+1}) \in \arg\max_{j,s} Gain_{R^{(l)}}(y, j, s).$$

This contradicts (A.8). Therefore, for any $l \in \{0, \ldots, L-2\}$, if $R^{(l)} \in \textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}$, then $R^{(l+1)} \in \textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}$. Since $R^{(0)} \in \textsc{tree}^{\lambda}\{y'(\phi, \nu_{sib})\}$, the proof follows by induction.

A.3.2 Proof of Lemma 2.2

Let $R_A \in \textsc{tree}^{\lambda}(y)$ with $\textsc{branch}\{R_A, \textsc{tree}^{\lambda}(y)\} = ((j_1, s_1, e_1), \ldots, (j_L, s_L, e_L))$ such that $R_A = \bigcap_{l=1}^{L} \chi_{j_l, s_l, e_l}$. Since Algorithm 2.1 creates regions by intersecting halfspaces and set intersections are invariant to the order of intersection, it follows that $R_A = \bigcap_{l=1}^{L} \chi_{j_l, s_l, e_l} \in \textsc{tree}^{\lambda}\{y'(\phi, \nu_{reg})\}$ if and only if there exists $\pi \in \Pi$ such that

$$\left\{ \bigcap_{l=1}^{l'} \chi_{j_{\pi(l)}, s_{\pi(l)}, e_{\pi(l)}} \right\}_{l'=1}^{L} \subseteq \textsc{tree}^{\lambda}\{y'(\phi, \nu_{reg})\}.$$

By Definitions 2.6 and 2.7,

$$\mathcal{R}(\pi[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}]) = \left\{ \bigcap_{l=1}^{l'} \chi_{j_{\pi(l)}, s_{\pi(l)}, e_{\pi(l)}} \right\}_{l'=1}^{L}.$$

Thus,

$$\begin{aligned} S^\lambda_{reg} &= \left\{\phi : R_A \in \textsc{tree}^\lambda\{y'(\phi, \nu_{reg})\}\right\} \\ &= \bigcup_{\pi \in \Pi} \left\{\phi : \mathcal{R}(\pi[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}]) \subseteq \textsc{tree}^\lambda \left\{y'\left(\phi, \nu_{reg}\right)\right\}\right\} \\ &= \bigcup_{\pi \in \Pi} S^\lambda \left(\pi \left[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}\right], \nu_{reg}\right), \end{aligned}$$

where the third equality follows from the definition of $S^\lambda(\mathcal{B}, \nu)$ in (2.18).

A.4 Proofs for Section 2.4.2

A.4.1 Proof of Proposition 2.4.1

Recall that $\mathcal{B} = ((j_1, s_1, e_1), \ldots, (j_L, s_L, e_L))$ and $\mathcal{R}(\mathcal{B}) = \{R^{(0)}, \ldots, R^{(L)}\}$. Recall from (2.20) that $S_{grow}(\mathcal{B}, \nu) = \{\phi : \mathcal{R}(\mathcal{B}) \subseteq \textsc{tree}^0\{y'(\phi, \nu)\}\}$, and that we define $S_{l,j,s} = \left\{\phi : \textsc{gain}_{R^{(l-1)}}\{y'(\phi, \nu), j, s\} \leq \textsc{gain}_{R^{(l-1)}}\{y'(\phi, \nu), j_l, s_l\}\right\}$.

For $l = 1, \ldots, L$,

$$R^{(l-1)} \in \textsc{tree}^0\{y'(\phi, \nu)\} \text{ and } \phi \in \cap_{s=1}^{n-1} \cap_{j=1}^{p} S_{l,j,s} \iff \{R^{(l-1)}, R^{(l)}\} \subseteq \textsc{tree}^0\{y'(\phi, \nu)\}, \quad \text{(A.9)}$$

because, given that $R^{(l-1)} \in \textsc{tree}^0\{y'(\phi, \nu)\}$, $R^{(l)} \in \textsc{tree}^0\{y'(\phi, \nu)\}$ if and only if $(j_l, s_l) \in \arg\max_{(j,s): s \in \{1,\ldots,n-1\}, j \in \{1,\ldots,p\}} \textsc{gain}_{R^{(l-1)}}\left(y'(\phi, \nu), j, s\right)$. Combining (A.9) with the fact that $\{\phi : R^{(0)} \in \textsc{tree}^0\{y'(\phi, \nu)\}\} = \mathbb{R}$ yields

$$\bigcap_{l=1}^{L} \bigcap_{j=1}^{p} \bigcap_{s=1}^{n-1} S_{l,j,s} = \bigcap_{l=1}^{L} \left\{\phi : \{R^{(l-1)}, R^{(l)}\} \subseteq \textsc{tree}^0\{y'(\phi, \nu)\}\right\} = \{\phi : \mathcal{R}(\mathcal{B}) \subseteq \textsc{tree}^0\{y'(\phi, \nu)\}\}.$$

A.4.2 Proof of Proposition 2.4.2

Given a region R, let $\mathbb{1}(R)$ denote the vector in $\mathbb{R}^n$ such that the ith element is $1_{(x_i \in R)}$. Let $\mathcal{P}_{\mathbb{1}(R)} = \mathbb{1}(R)\left\{\mathbb{1}(R)^{\mathrm{T}}\mathbb{1}(R)\right\}^{-1}\mathbb{1}(R)^{\mathrm{T}}$ denote the orthogonal projection matrix onto the vector $\mathbb{1}(R)$.

Lemma A.4 *For any region R, $\textsc{gain}_R(y, j, s) = y^{\mathrm{T}} M_{R,j,s} y$, where*

$$M_{R,j,s} = \mathcal{P}_{\mathbb{1}(R\cap\chi_{j,s,1})} + \mathcal{P}_{\mathbb{1}(R\cap\chi_{j,s,0})} - \mathcal{P}_{\mathbb{1}(R)}. \tag{A.10}$$

Furthermore, the matrix $M_{R,j,s}$ is positive semidefinite.

A.4.2.0.1 Proof: For any region R, $\sum_{i\in R}(y_i - \bar{y}_R)^2 = \sum_{i\in R} y_i^2 - y^{\mathrm{T}}\mathcal{P}_{\mathbb{1}_{(R)}} y$. Thus, from (2.2),

$$\begin{aligned}
\textsc{gain}_R(y, j, s) &= \sum_{i\in R}(y_i - \bar{y}_R)^2 - \sum_{i\in R\cap\chi_{j,s,1}}(y_i - \bar{y}_{R\cap\chi_{j,s,1}})^2 - \sum_{i\in R\cap\chi_{j,s,0}}(y_i - \bar{y}_{R\cap\chi_{j,s,0}})^2 \\
&= \sum_{i\in R} y_i^2 - y^{\mathrm{T}}\mathcal{P}_{\mathbb{1}_{(R)}} y - \sum_{i\in R\cap\chi_{j,s,1}} y_i^2 + y^{\mathrm{T}}\mathcal{P}_{\mathbb{1}(R\cap\chi_{j,s,1})} y - \sum_{i\in R\cap\chi_{j,s,0}} y_i^2 + y^{\mathrm{T}}\mathcal{P}_{\mathbb{1}(R\cap\chi_{j,s,0})} y \\
&= y^{\mathrm{T}}\left\{\mathcal{P}_{\mathbb{1}(R\cap\chi_{j,s,1})} + \mathcal{P}_{\mathbb{1}(R\cap\chi_{j,s,0})} - \mathcal{P}_{\mathbb{1}(R)}\right\} y = y^{\mathrm{T}} M_{R,j,s} y.
\end{aligned}$$

To see that $M_{R,j,s}$ is positive semidefinite, observe that, for any vector v,

$$\begin{aligned}
v^{\mathrm{T}} M_{R,j,s} v = \textsc{gain}_R(v, j, s) &= \sum_{i\in R}(v_i - \bar{v}_R)^2 - \min_{a_1,a_2}\left\{\sum_{i\in R\cap\chi_{j,s,1}}(v_i - a_1)^2 + \sum_{i\in R\cap\chi_{j,s,0}}(v_i - a_2)^2\right\} \\
&\geq \sum_{i\in R}(v_i - \bar{v}_R)^2 - \left\{\sum_{i\in R\cap\chi_{j,s,1}}(v_i - \bar{v}_R)^2 + \sum_{i\in R\cap\chi_{j,s,0}}(v_i - \bar{v}_R)^2\right\} = 0.
\end{aligned}$$

□

It follows from Lemma A.4 that we can express each set $S_{l,j,s}$ from Proposition 2.4.1 as

$$\begin{aligned} S_{l,j,s} &= \{\phi : \text{GAIN}_{R^{(l-1)}}\{y'(\phi,\nu), j, s\} \leq \text{GAIN}_{R^{(l-1)}}\{y'(\phi,\nu), j_l, s_l\}\} \\ &= \left\{\phi : y'(\phi,\nu)^{\mathrm{T}} M_{R^{(l-1)},j,s} y'(\phi,\nu) \leq y'(\phi,\nu)^{\mathrm{T}} M_{R^{(l-1)},j_l,s_l} y'(\phi,\nu)\right\}. \end{aligned} \tag{A.11}$$

We now use (A.11) to prove the first statement of Proposition 2.4.2.

Lemma A.5 *Each set $S_{l,j,s}$ is defined by a quadratic inequality in ϕ.*

A.4.2.0.2 Proof: The definition of $y'(\phi,\nu)$ in (2.10) implies that

$$\begin{aligned} y'(\phi,\nu)^{\mathrm{T}} M_{R,j,s} y'(\phi,\nu) &= \left(\mathcal{P}_\nu^\perp y + \frac{\nu\phi}{\|\nu\|_2^2}\right)^{\mathrm{T}} M_{R,j,s} \left(\mathcal{P}_\nu^\perp y + \frac{\nu\phi}{\|\nu\|_2^2}\right) \\ &= \frac{\nu^{\mathrm{T}} M_{R,j,s}\nu}{\|\nu\|_2^4}\phi^2 + \frac{2\nu^{\mathrm{T}} M_{R,j,s}\mathcal{P}_\nu^\perp y}{\|\nu\|_2^2}\phi + y^{\mathrm{T}} P_\nu^\perp M_{R,j,s} P_\nu^\perp y \\ &\equiv a(R,j,s)\phi^2 + b(R,j,s)\phi + c(R,j,s). \end{aligned} \tag{A.12}$$

Therefore, by (A.11),

$$\begin{aligned} S_{l,j,s} = \Big\{\phi : &\left[a\left\{R^{(l-1)}, j, s\right\} - a\left\{R^{(l-1)}, j_l, s_l\right\}\right]\phi^2 + \left[b\left\{R^{(l-1)}, j, s\right\} - b\left\{R^{(l-1)}, j_l, s_l\right\}\right]\phi \\ &+ \left[c\left\{R^{(l-1)}, j, s\right\} - c\left\{R^{(l-1)}, j_l, s_l\right\}\right] \leq 0\Big\}. \end{aligned} \tag{A.13}$$

□

Proposition 2.4.1 indicates that to compute $S_{grow}(\mathcal{B},\nu)$ from (2.20), we need to compute the coefficients of the quadratic for each $S_{l,j,s}$, where $l = 1,\ldots,L$, $j = 1,\ldots,p$, and $s = 1,\ldots,n-1$.

Lemma A.6 *We can compute the coefficients $a\left\{R^{(l-1)}, j, s\right\}, b\left\{R^{(l-1)}, j, s\right\}$ and $c\left\{R^{(l-1)}, j, s\right\}$, defined in Lemma A.5, for $l = 1,\ldots,L$, $j = 1,\ldots,p$, and $s = 1,\ldots,n-1$, in $O\{nplog(n) + npL\}$ operations.*

A.4.2.0.3 Proof: Using the definitions in Lemmas A.4 and A.5 and algebra, we have that

$$\|\nu\|_2^4 a\{R^{(l-1)},j,s\} = \frac{\left[\nu^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,1}\}\right]^2}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,1}\}}} + \frac{\left[\nu^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,0}\}\right]^2}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,0}\}}} - \frac{\left[\nu^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\}\right]^2}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\}}}, \tag{A.14}$$

$$\frac{1}{2}\|\nu\|_2^2 b\{R^{(l-1)},j,s\} = \frac{\nu^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,1}\}\left(\mathcal{P}_\nu^\perp y\right)^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,1}\}}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,1}\}}} + \frac{\nu^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,0}\}\left(\mathcal{P}_\nu^\perp y\right)^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,0}\}}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,0}\}}} - \frac{\nu^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\}\left(\mathcal{P}_\nu^\perp y\right)^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\}}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\}}}, \tag{A.15}$$

$$c\{R^{(l-1)},j,s\} = \frac{\left[(\mathcal{P}_\nu^\perp y)^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,1}\}\right]^2}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,1}\}}} + \frac{\left[(\mathcal{P}_\nu^\perp y)^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\cap\chi_{j,s,0}\}\right]^2}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,0}\}}} - \frac{\left[(\mathcal{P}_\nu^\perp y)^{\mathrm{T}}\mathbb{1}\{R^{(l-1)}\}\right]^2}{\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\}}}. \tag{A.16}$$

We compute the scalar $\|\nu\|_2^2$ and the vector $\mathcal{P}_\nu^\perp y$ in $O(n)$ operations once at the start of the algorithm. We also sort each feature in $O\left[nlog(n)\right]$ operations per feature. We will now show that for the lth level and the jth feature, we can compute $a\{R^{(l-1)},j,s\}$, $b\{R^{(l-1)},j,s\}$, and $c\{R^{(l-1)},j,s\}$ for all $n-1$ values of s in $O(n)$ operations.

The index s appears in (A.14)–(A.16) only through $\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,1}\}}$, $\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,0}\}}$, and through inner products of vectors ν and $\mathcal{P}_\nu^\perp y$ with indicator vectors $\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,s,1}\right\}$ and $\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,s,0}\right\}$. For simplicity, we assume that covariate x_j is continuous, and thus the order statistics are unique.

Let m_1 be the index corresponding to the smallest value of x_j. Then $\nu^{\mathrm{T}}\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,1,1}\right\} = \nu_{m_1}$ if observation m_1 is in $R^{(l-1)}$, and is 0 otherwise. Similarly, $\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,1,1}\}} = 1$ if observation m_1 is in $R^{(l-1)}$, and is 0 otherwise. Next, let m_2 be the index corresponding to the second smallest value of x_j. Then $\nu^{\mathrm{T}}\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,2,1}\right\} = \mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,1,1}\right\} + \nu_{m_2}$ if observation m_2 is in $R^{(l-1)}$, and is equal to $\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,1,1}\right\}$ otherwise. We compute $\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,2,1}\}}$ in the same manner. Each update is done in constant time. Continuing in this manner, computing the full set of $n-1$ quantities $\nu^{\mathrm{T}}\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,s,1}\right\}$ and $\sum_{i=1}^n 1_{\{i\in R^{(l-1)}\cap\chi_{j,s,1}\}}$ for $s=1,\ldots,n-1$ requires a single forward pass through the sorted values of x_j, which takes $O(n)$ operations. The same ideas can be applied to compute $\left(\mathcal{P}_\nu^\perp y\right)^{\mathrm{T}}\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,1,1}\right\}$, $\nu^{\mathrm{T}}\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,s,0}\right\}$, $\left(\mathcal{P}_\nu^\perp y\right)^{\mathrm{T}}\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j,s,0}\right\}$,

and $\sum_{i=1}^{n} 1_{\{i \in R^{(l-1)} \cap \chi_{j,s,0}\}}$ using constant time updates for each value of s.

Thus, we can obtain all components of coefficients $a\{R^{(l-1)}, j, s\}$, $b\{R^{(l-1)}, j, s\}$, and $c\{R^{(l-1)}, j, s\}$ for a fixed j and l, and for all $s = 1, \ldots, n-1$, in $O(n)$ operations. These scalar components can be combined to obtain the coefficients in $O(n)$ operations. Therefore, given the sorted features, we compute the $(n-1)pL$ coefficients in $O(npL)$ operations. □

Once the coefficients on the right hand side of (A.13) have been computed, we can compute $S_{l,j,s}$ in constant time via the quadratic equation: it is either a single interval or the union of two intervals. Finally, in general we can intersect $(n-1)pL$ intervals in $O\{npL \times log(npL)\}$ operations [Bourgon, 2009]. The final claim of Proposition 2.4.2 involves the special case where $\nu = \nu_{sib}$ and $\mathcal{B} = \text{BRANCH}\{R_A, \text{TREE}^{\lambda}(y)\}$.

Lemma A.7 *Suppose that $\nu = \nu_{sib}$ from (2.6) and $\mathcal{B} = \text{BRANCH}\{R_A, \text{TREE}^{\lambda}(y)\}$. (i) If $l < L$, then for all j and s, there exist $a, b \in [-\infty, \infty]$ such that $a \leq \nu^{\mathrm{T}} y \leq b$ and $S_{l,j,s} = (a, b)$. (ii) If $l = L$, then for all j and s, there exist $c, d \in \mathbb{R}$ such that $c \leq 0 \leq d$ and $S_{l,j,s} = (-\infty, c] \cup [d, \infty)$. (iii) We can intersect all $(n-1)pL$ sets of the form $S_{l,j,s}$ in $O(npL)$ operations.*

A.4.2.0.4 Proof: This proof relies on the form of $S_{l,j,s}$ given in (A.13). To prove (i), note that when $l < L$,

$$\begin{aligned}\|\nu_{sib}\|_2^4 \left[a\left\{R^{(l-1)}, j, s\right\} - a\left\{R^{(l-1)}, j_l, s_l\right\}\right] &= \nu_{sib}^{\mathrm{T}} M_{R^{(l-1)},j,s} \nu_{sib} - \nu_{sib}^{\mathrm{T}} M_{R^{(l-1)},j_l,s_l} \nu_{sib} \\ &= \nu_{sib}^{\mathrm{T}} M_{R^{(l-1)},j,s} \nu_{sib} \geq 0.\end{aligned}$$

The first equality follows directly from the definition of $a(R, j, s)$ in (A.12). To see why the second equality holds, observe that $R_A \cup R_B \subseteq R^{(l-1)}$, and without loss of generality assume that $R_A \cup R_B \subseteq R^{(l-1)} \cap \chi_{j_l,s_l,1}$. Recall that the ith element of ν_{sib} is non-zero if and only if $i \in R_A \cup R_B$, and that the non-zero elements of ν_{sib} sum to 0. Thus, $\mathbb{1}\left\{R^{(l-1)}\right\}^{\mathrm{T}} \nu_{sib} = 0$ and $\mathbb{1}\left\{R^{(l-1)} \cap \chi_{j_l,s_l,1}\right\}^{\mathrm{T}} \nu_{sib} = 0$. Furthermore, the supports of $R^{(l-1)} \cap \chi_{j_l,s_l,0}$ and ν_{sib} are

non-overlapping, and so $\mathbb{1}\left\{R^{(l-1)} \cap \chi_{j_l,s_l,0}\right\}^{\mathrm{T}} \nu_{sib} = 0$. Thus,

$$M_{R^{(l-1)},j_l,s_l}\nu_{sib} = \left[\mathcal{P}_{\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j_l,s_l,1}\right\}} + \mathcal{P}_{\mathbb{1}\left\{R^{(l-1)}\cap\chi_{j_l,s_l,0}\right\}} - \mathcal{P}_{\mathbb{1}\left\{R^{(l-1)}\right\}}\right]\nu_{sib} = 0.$$

The final inequality follows because $M_{R^{(l-1)},j,s}$ is positive semidefinite (Lemma A.4).

Thus, when $l < L$, $S_{l,j,s}$ is defined in (A.13) by a quadratic inequality with a non-negative quadratic coefficient. Thus, $S_{l,j,s}$ must be a single interval of the form (a, b). Furthermore, since $\mathcal{B} = \textsc{branch}\{R_A, \textsc{tree}^{\lambda}(y)\}$, we know that $\mathcal{R}(\mathcal{B}) \subseteq \textsc{tree}^0(y) = \textsc{tree}^0\{y'(\nu^{\mathrm{T}}y, \nu)\}$. Therefore, $\nu^{\mathrm{T}}y \in S_{grow}(\mathcal{B}, \nu_{sib}) = \cap_{l=1}^{L}\cap_{j=1}^{p}\cap_{s=1}^{n-1} S_{l,j,s}$, and so we conclude $a \leq \nu^{\mathrm{T}}y \leq b$. This completes the proof of (i).

To prove (ii), we first prove that when $l = L$ the quadratic equation in ϕ defined in (A.13) has a non-positive quadratic coefficient. To see this, note that

$$\begin{aligned}
&\|\nu_{sib}\|_2^4\left[a\left\{R^{(L-1)},j,s\right\} - a\left\{R^{(L-1)},j_L,s_L\right\}\right] = \nu_{sib}^{\mathrm{T}}M_{R^{(L-1)},j,s}\nu_{sib} - \nu_{sib}^{\mathrm{T}}M_{R^{(L-1)},j_L,s_L}\nu_{sib}\\
&= \nu_{sib}^{\mathrm{T}}\left[\mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,1}\}} + \mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,0}\}}\right]\nu_{sib} - \nu_{sib}^{\mathrm{T}}\left[\mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j_L,s_L,1}\}} + \mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j_L,s_L,0}\}}\right]\nu_{sib}\\
&= \nu_{sib}^{\mathrm{T}}\left[\mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,1}\}} + \mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,0}\}}\right]\nu_{sib} - \nu_{sib}^{\mathrm{T}}\nu_{sib}\\
&= \left\|\left[\mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,1}\}} + \mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,0}\}}\right]\nu_{sib}\right\|_2^2 - \|\nu_{sib}\|_2^2 \leq 0. \qquad \text{(A.17)}
\end{aligned}$$

The first equality follows from (A.12). The second follows from the definition of $M_{R,j,s}$ given in (A.10) and from the fact that $\mathcal{P}_{\mathbb{1}\{R^{(L-1)}\}}\nu_{sib} = 0$ because $\mathbb{1}\{R^{(L-1)}\}^{\mathrm{T}}\nu_{sib}$ sums up all of the non-zero elements of ν_{sib}, which sum to 0. The third equality follows because ν_{sib} lies in $span\left[\mathbb{1}\left\{R^{(L-1)}\cap\chi_{j_L,s_L,1}\right\}, \mathbb{1}\left\{R^{(L-1)}\cap\chi_{j_L,s_L,0}\right\}\right]$; projecting it onto this span yields itself. Noting that $\left[\mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,1}\}} + \mathcal{P}_{\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,0}\}}\right]$ is itself a projection matrix, the fourth equality follows from the idempotence of projection matrices, and the inequality follows from the fact that $\|\nu_{sib}\|_2 \geq \|Q\nu_{sib}\|_2$ for any projection matrix Q. Thus, when $l = L$, the quadratic that defines $S_{l,j,s}$ has a non-positive quadratic coefficient.

Equality is attained in (A.17) if and only if $\nu_{sib} \in span\left[\mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,1}\}, \mathbb{1}\{R^{(L-1)}\cap\chi_{j,s,0}\}\right]$. This can only happen if splitting $R^{(L-1)}$

on j, s yields an identical partition of the data to splitting on j_L, s_L. If this is the case, then $S_{L,j,s} = (-\infty, 0] \cup [0, \infty)$ from the definition of $S_{l,j,s}$ in Proposition 2.4.1, and so (ii) is satisfied with $c = d = 0$.

We now proceed to the setting where the inequality in (A.17) is strict. In this case, (A.13) implies that $S_{L,j,s} = (-\infty, c] \cup [d, \infty)$ for $c \leq d$ and $c, d \in \mathbb{R}$. To complete the proof of (ii), we must argue that $c \leq 0$ and $d \geq 0$. Recall that the quadratic in (A.11) has the form $\textsc{gain}_{R^{(L-1)}}\{y'(\phi, \nu), j, s\} - \textsc{gain}_{R^{(L-1)}}\{y'(\phi, \nu), j_L, s_L\}$. When $\phi = 0$, $\textsc{gain}_{R^{(L-1)}}\{y'(\phi, \nu), j_L, s_L\} = 0$, because $\phi = 0$ eliminates the contrast between R_A and R_B, so that the split on j_L, s_L provides zero gain. So, when $\phi = 0$, the quadratic evaluates to $\textsc{gain}_{R^{(L-1)}}\{y'(\phi, \nu), j, s\}$, which is non-negative by Lemma A.4. Thus, $S_{l,j,s}$ is defined by a downward facing quadratic that is non-negative when $\phi = 0$, and so the set $S_{l,j,s}$ has the form $(-\infty, c] \cup [d, \infty)$ for $c \leq 0 \leq d$.

To prove (iii), observe that (i) implies that $\cap_{l=1}^{L-1} \cap_{j=1}^{p} \cap_{s=1}^{n-1} S_{l,j,s} = (a_{max}, b_{min})$, where a_{max} is the maximum over all of the a's, and b_{min} is the minimum over all of the b's. This can be computed in $np(L-1)$ steps. Furthermore, (ii) implies that $\cap_{j=1}^{p} \cap_{s=1}^{n-1} S_{L,j,s} = (-\infty, c_{min}] \cap [d_{max}, \infty)$, where c_{min} and d_{max} are the minimum over all of the c's and the maximum over all the d's, respectively. This can be computed in np steps. Thus, we can compute $\cap_{l=1}^{L} \cap_{j=1}^{p} \cap_{s=1}^{n-1} S_{l,j,s}$ in $O(npL)$ operations. □

A.4.3 Proof of Proposition 2.4.3

To prove Proposition 2.4.3, we first propose a particular method of constructing an example of $\textsc{tree}(\mathcal{B}, \nu, \lambda)$. We then show that (2.21) holds for this particular choice for $\textsc{tree}(\mathcal{B}, \nu, \lambda)$. We conclude by evaluating the computational cost of computing such an example of $\textsc{tree}(\mathcal{B}, \nu, \lambda)$, and by arguing that in the special case where $\mathcal{R}(\mathcal{B}) \in \textsc{tree}^{\lambda}(y)$, our example is equal to $\textsc{tree}^{\lambda}(y)$.

When Algorithm 2.2 is called with parameters $\textsc{tree}, y, \lambda, \mathcal{O}$, where $\mathcal{O}$ is a bottom-up ordering of the K nodes in $\textsc{tree}$, it computes a sequence of intermediate trees, $\textsc{tree}_0, \ldots, \textsc{tree}_K$. We use the notation $\textsc{tree}_k(\textsc{tree}, y, \lambda, \mathcal{O})$, for $k = 0, \ldots, K$, to denote

the kth of these intermediate trees. The following lemma helps build up to our proposed example of $\textsc{tree}(\mathcal{B}, \nu, \lambda)$.

Lemma A.8 *Let* $\phi_1 \in S_{grow}(\mathcal{B}, \nu)$ *and* $\phi_2 \in S_{grow}(\mathcal{B}, \nu)$. *Then* $\textsc{tree}^0\{y'(\phi_1, \nu)\} = \textsc{tree}^0\{y'(\phi_2, \nu)\}$. *Let* $\mathcal{O}$ *be a bottom-up ordering of the* K *regions in* $\textsc{tree}^0\{y'(\phi_1, \nu)\}$ *such that the last* L *regions in the ordering are* $R^{(L-1)}, \ldots, R^{(0)}$. *Then* $\textsc{tree}_{K-L}[\textsc{tree}^0\{y'(\phi_1, \nu)\}, y'(\phi_1, \nu), \lambda, \mathcal{O}] = \textsc{tree}_{K-L}[\textsc{tree}^0\{y'(\phi_2, \nu)\}, y'(\phi_2, \nu), \lambda, \mathcal{O}]$.

A.4.3.0.1 Proof: We first prove that $\textsc{tree}^0\{y'(\phi_1, \nu)\} \subseteq \textsc{tree}^0\{y'(\phi_2, \nu)\}$, where $\phi_1, \phi_2 \in S_{grow}(\mathcal{B}, \nu)$. The fact that $\phi_1 \in S_{grow}(\mathcal{B}, \nu)$ and $\phi_2 \in S_{grow}(\mathcal{B}, \nu)$ implies two properties:

> *Property 1:* $R^{(l)} \in \textsc{tree}^0\{y'(\phi_1, \nu)\}$ and $R^{(l)} \in \textsc{tree}^0\{y'(\phi_2, \nu)\}$ for $l \in \{0, \ldots, L\}$ by the definition of $S_{grow}(\mathcal{B}, \nu)$.
>
> *Property 2:* $R^{(l)}_{sib} \in \textsc{tree}^0\{y'(\phi_1, \nu)\}$ and $R^{(l)}_{sib} \in \textsc{tree}^0\{y'(\phi_2, \nu)\}$ for $l \in \{1, \ldots, L\}$, where $R^{(l)}_{sib} \equiv R^{(l-1)} \cap \chi_{j_l, s_l, 1-e_l}$. This follows from Property 1 and Definition 2.1.

Suppose that $R \in \textsc{tree}^0\{y'(\phi_1, \nu)\}$. Then R must belong to one of these three cases, illustrated in Figure A.1(a):

> *Case 1:* $\exists\ l \in \{0, \ldots L\}$ *such that* $R = R^{(l)}$. By Property 1, $R \in \textsc{tree}^0\{y'(\phi_2, \nu)\}$.
>
> *Case 2:* $\exists\ l \in \{1, \ldots, L\}$ *such that* $R = R^{(l)}_{sib}$. By Property 2, $R \in \textsc{tree}^0\{y'(\phi_2, \nu)\}$.
>
> *Case 3:* $R \in \textsc{desc}[R', \textsc{tree}^0\{y'(\phi_1, \nu)\}]$, *where either* $R' = R^{(l)}_{sib}$ *for some* $l \in \{1, \ldots, L\}$, *or else* $R' = R^{(L)}$. By Properties 1 and 2, $R' \in \textsc{tree}^0\{y'(\phi_2, \nu)\}$. Condition 2.1 ensures that, for all $i \in R'$ and for some constants c and d,

$$\{y'(\phi_2, \nu)\}_i = \begin{cases} \{y'(\phi_1, \nu)\}_i & \text{if } R' = R^{(l)}_{sib} \text{ for some } l \in \{1, \ldots, L-1\}, \\ \{y'(\phi_1, \nu)\}_i + c & \text{if } R' = R^{(L)}_{sib}, \\ \{y'(\phi_1, \nu)\}_i + d & \text{if } R' = R^{(L)}. \end{cases}$$

As constant shifts preserve within-node sums of squared errors, in each of these three scenarios, $\textsc{desc}[R', \textsc{tree}^0\{y'(\phi_1,\nu)\}] = \textsc{desc}[R', \textsc{tree}^0\{y'(\phi_2,\nu)\}]$. Thus, $R \in \textsc{tree}^0\{y'(\phi_2,\nu)\}$.

Thus, if $R \in \textsc{tree}^0\{y'(\phi_1,\nu)\}$, then $R \in \textsc{tree}^0\{y'(\phi_2,\nu)\}$. This completes the argument that $\textsc{tree}^0\{y'(\phi_1,\nu)\} \subseteq \textsc{tree}^0\{y'(\phi_2,\nu)\}$. Swapping the roles of ϕ_1 and ϕ_2 in this argument, we see that $\textsc{tree}^0\{y'(\phi_2,\nu)\} \subseteq \textsc{tree}^0\{y'(\phi_1,\nu)\}$. This concludes the proof that $\textsc{tree}^0\{y'(\phi_1,\nu)\} = \textsc{tree}^0\{y'(\phi_2,\nu)\}$.

Because $\textsc{tree}^0\{y'(\phi_1,\nu)\} = \textsc{tree}^0\{y'(\phi_2,\nu)\}$, it follows that any bottom-up ordering of the regions in $\textsc{tree}^0\{y'(\phi_1,\nu)\}$ is also a bottom-up ordering for the regions in $\textsc{tree}^0\{y'(\phi_2,\nu)\}$. We next prove by induction that, if we choose a bottom-up ordering $\mathcal{O}$ that places the regions in $\mathcal{R}(\mathcal{B})$ at the end of the ordering, then

$$\textsc{tree}_k[\textsc{tree}^0\{y'(\phi_1,\nu)\}, y'(\phi_1,\nu), \lambda, \mathcal{O}] = \textsc{tree}_k[\textsc{tree}^0\{y'(\phi_2,\nu)\}, y'(\phi_2,\nu), \lambda, \mathcal{O}], \tag{A.18}$$

for $k = 0, \ldots, K-L$. It follows immediately from Algorithm 2.2 and the argument above that $\textsc{tree}_0[\textsc{tree}^0\{y'(\phi_1,\nu)\}, y'(\phi_1,\nu), \lambda, \mathcal{O}] = \textsc{tree}_0[\textsc{tree}^0\{y'(\phi_2,\nu)\}, y'(\phi_2,\nu), \lambda, \mathcal{O}]$. Next, suppose that for some $k \in \{1, \ldots, K-L\}$,

$$\textsc{tree}_{k-1}[\textsc{tree}^0\{y'(\phi_1,\nu)\}, y'(\phi_1,\nu), \lambda, \mathcal{O}] = \textsc{tree}_{k-1}[\textsc{tree}^0\{y'(\phi_2,\nu)\}, y'(\phi_2,\nu), \lambda, \mathcal{O}], \tag{A.19}$$

and denote this tree with $\textsc{tree}_{k-1}$ for brevity. We must prove that (A.18) holds. Let R be the kth region in $\mathcal{O}$ and recall the assumption that the last L regions in $\mathcal{O}$ are $\{R^{(L-1)}, \ldots, R^{(0)}\}$. Since $k \leq K-L$, this implies that $R \notin \{R^{(L-1)}, \ldots, R^{(0)}\}$. This means that either $R \in \textsc{desc}\left[R^{(l)}_{sib}, \textsc{tree}^0\{y'(\phi_1,\nu)\}\right]$ for $l \in \{1, \ldots, L\}$ or $R \in \textsc{desc}\left[R^{(L)}, \textsc{tree}^0\{y'(\phi_1,\nu)\}\right]$, meaning that R is a black region in Figure A.1(b). From Condition 2.1,

$$\{y'(\phi_2,\nu)\}_i = \begin{cases} \{y'(\phi_1,\nu)\}_i & \text{if } R \in \textsc{desc}\left[R^{(l)}_{sib}, \textsc{tree}^0\{y'(\phi_1,\nu)\}\right] \text{ for } l \in \{1, \ldots, L-1\}, \\ \{y'(\phi_1,\nu)\}_i + c & \text{if } R \in \textsc{desc}\left[R^{(L)}_{sib}, \textsc{tree}^0\{y'(\phi_1,\nu)\}\right], \\ \{y'(\phi_1,\nu)\}_i + d & \text{if } R \in \textsc{desc}\left[R^{(L)}, \textsc{tree}^0\{y'(\phi_1,\nu)\}\right]. \end{cases}$$

In any of the three cases illustrated in Figure A.1, for $g(\cdot)$ defined in (2.3), $g\{R, \text{TREE}_{k-1}, y'(\phi_1, \nu)\} = g\{R, \text{TREE}_{k-1}, y'(\phi_2, \nu)\}$. Combining this with (A.19) and Step 2(b) of Algorithm 2.2 yields (A.18). This completes the proof by induction.

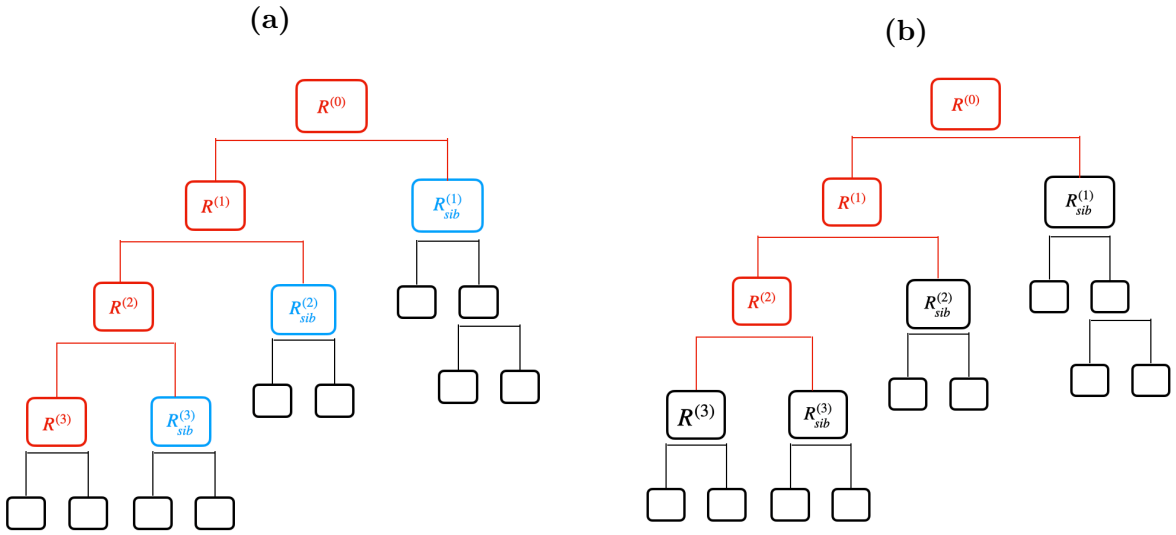


Figure A.1: *(a).* An illustration of Case 1 (red), Case 2 (blue), and Case 3 (black) for a region $R \in \text{TREE}^0\{y'(\phi_1, \nu)\}$ in the base case of the proof of Lemma A.8, where $\mathcal{R}(\mathcal{B}) = \{R^{(0)}, \ldots, R^{(3)}\}$. *(b.)* The black regions show the possible cases for $R \in \text{TREE}_{k-1}$ in the inductive step of the proof of Lemma A.8.

□

Since Lemma A.8 guarantees that each $\phi \in S_{grow}(\mathcal{B}, \nu)$ leads to the same $\text{TREE}^0\{y'(\phi, \nu)\}$, we will refer to this tree as TREE^0, will let K be the number of regions in this tree, and will let $\mathcal{O}$ be a bottom-up ordering of these regions that places $R^{(L-1)}, \ldots, R^{(0)}$ in the last L spots. We will further denote $\text{TREE}_{K-L}\{\text{TREE}^0, y'(\phi, \nu), \lambda, \mathcal{O}\}$ for any $\phi \in S_{grow}(\mathcal{B}, \nu)$ as TREE_{K-L}, since Lemma A.8 further tells us that this is the same for all $\phi \in S_{grow}(\mathcal{B}, \nu)$. In what follows, we argue that if we let $\text{TREE}(\mathcal{B}, \nu, \lambda) = \text{TREE}_{K-L}$, where $\text{TREE}(\mathcal{B}, \nu, \lambda)$ appears in the statement of Proposition 2.4.3, then (2.21) holds. In other words, we prove that TREE_{K-L}, which always exists and is well-defined, is a valid example of $\text{TREE}(\mathcal{B}, \nu, \lambda)$.

Recall from (2.18) that $S^{\lambda}(\mathcal{B}, \nu) = \big\{\phi \in S_{grow}(\mathcal{B}, \nu) : R^{(L)} \in \text{TREE}^{\lambda}\{y'(\phi, \nu)\}\big\}$.

Lemma A.8 says that for $\phi \in S_{grow}(\mathcal{B}, \nu)$, we can rewrite $\textsc{tree}^{\lambda}\{y'(\phi, \nu)\}$ as $\textsc{tree}_K\{\textsc{tree}^0, y'(\phi, \nu), \lambda, \mathcal{O}\}$. So we can rewrite $S^{\lambda}(\mathcal{B}, \nu)$ as

$$S^{\lambda}(\mathcal{B}, \nu) = \left\{\phi \in S_{grow}(\mathcal{B}, \nu) : R^{(L)} \in \textsc{tree}_K\left\{\textsc{tree}^0, y'(\phi, \nu), \lambda, \mathcal{O}\right\}\right\}. \tag{A.20}$$

Furthermore, since $R^{(L-1)}, \ldots, R^{(0)}$ (all of which are ancestors of $R^{(L)}$) are the last L nodes in the ordering $\mathcal{O}$, we see that $R^{(L)} \in \textsc{tree}_K\{\textsc{tree}^0, y'(\phi, \nu), \lambda, \mathcal{O}\}$ if and only if no pruning occurs during the last L iterations of Step 2 in Algorithm 2.2. This means that we can characterize (A.20) as

$$\left\{\phi \in S_{grow}(\mathcal{B}, \nu) : \textsc{tree}_{K-L}\{\textsc{tree}^0, y'(\phi, \nu), \lambda, \mathcal{O}\} = \textsc{tree}_K\left\{\textsc{tree}^0, y'(\phi, \nu), \lambda, \mathcal{O}\right\}\right\}. \tag{A.21}$$

Recall that for $k = K - L + 1, \ldots, K$, $R^{(K-k)}$ is the kth region in $\mathcal{O}$, and is an ancestor of $R^{(L)}$. We next argue that we can rewrite (A.21) as

$$\bigcap_{k=K-L+1}^{K}\left\{\phi \in S_{grow}(\mathcal{B}, \nu) : g\left\{R^{(K-k)}, \textsc{tree}_{K-L}, y'(\phi, \nu)\right\} \geq \lambda\right\}. \tag{A.22}$$

To begin, suppose that $\phi \in$ (A.22). As we are talking about a particular ϕ, for $k = 0, \ldots, K$ we will suppress the dependence of $\textsc{tree}_k\left\{\textsc{tree}^0, y'(\phi, \nu), \lambda, \mathcal{O}\right\}$ on its arguments and denote it with $\textsc{tree}_k$. The fact that $\phi \in$ (A.22) means that $g\left\{R^{(L-1)}, \textsc{tree}_{K-L}, y'(\phi, \nu)\right\} \geq \lambda$, which ensures that no pruning occurs at step $K - L + 1$, which in turn ensures that $\textsc{tree}_{K-L+1} = \textsc{tree}_{K-L}$. Combined with (A.22), this implies that $g\left\{R^{(L-2)}, \textsc{tree}_{K-L}, y'(\phi, \nu)\right\} =$
$g\left\{R^{(L-2)}, \textsc{tree}_{K-L+1}, y'(\phi, \nu)\right\} \geq \lambda$, which ensures that no pruning occurs at step $K-L+2$, which in turn ensures that $\textsc{tree}_{K-L+2} = \textsc{tree}_{K-L+1} = \textsc{tree}_{K-L}$. Proceeding in this manner, by tracing through the last L iterations of Step 2 of Algorithm 2.2, we see that ϕ satisfies $\textsc{tree}_K = \textsc{tree}_{K-L}$, and so $\phi \in$ (A.21).

Next suppose that $\phi \notin$ (A.22). Let

$$k' = \min_{k \in \{K-L+1, \ldots, K\}}\left\{k : g\left\{R^{(K-k)}, \textsc{tree}_{K-L}, y'(\phi, \nu)\right\} < \lambda\right\}.$$

As k' is a minimum, we know that no pruning occurred during steps $K-L+1,\ldots,k'-1$, and so $\text{TREE}_{k'-1}\left\{\text{TREE}^0(y'(\phi,\nu)), y'(\phi,\nu), \lambda, \mathcal{O}\right\} = \text{TREE}_{K-L}$. This implies that $g\left\{R^{(K-k')}, \text{TREE}_{K-L}, y'(\phi,\nu)\right\} < \lambda$ can be rewritten as $g\left(R^{(K-k')}, \text{TREE}_{k'-1}\left[\text{TREE}^0\{y'(\phi,\nu)\}, y'(\phi,\nu), \lambda, \mathcal{O}\right], y'(\phi,\nu)\right) < \lambda$. It then follows from Algorithm 2.2 that pruning occurs at step k', which means that TREE_K cannot possibly equal TREE_{K-L}. Thus, $\phi \notin$ (A.21).

Thus, $\phi \in$ (A.22) if and only if $\phi \in$ (A.21).

Finally, Proposition 2.4.3 rewrites (A.22) with the indexing over k changed to an indexing over l, and plugging in $\text{TREE}(\mathcal{B},\nu,\lambda) = \text{TREE}_{K-L}$. Therefore, TREE_{K-L} is a valid example of $\text{TREE}(\mathcal{B},\nu,\lambda)$.

To compute $\text{TREE}(\mathcal{B},\nu,\lambda)$, we first select an arbitrary $\phi \in S_{grow}(\mathcal{B},\nu)$. We then apply Algorithm 2.1 to grow $\text{TREE}^0\{y'(\phi,\nu)\}$. We create a bottom-up ordering $\mathcal{O}$ of the K nodes in $\text{TREE}^0\{y'(\phi,\nu)\}$ such that $R^{(L-1)},\ldots,R^{(0)}$ are at the end. Finally, we apply the first $K-L$ iterations of Algorithm 2.2 with arguments $\text{TREE}^0\{y'(\phi,\nu)\}, y'(\phi,\nu), \lambda$, and $\mathcal{O}$ to obtain $\text{TREE}(\mathcal{B},\nu,\lambda)$. The worst case computational cost of CART (the combined Algorithm 2.1 and Algorithm 2.2) is $O(n^2p)$.

In the special case that $\mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^\lambda(y)$, we have that $\nu^T y \in S_{grow}(\mathcal{B},\nu)$ because $y = y'(\nu^T y,\nu)$ and $\mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^\lambda(y) \subseteq \text{TREE}^0(y)$. In this case, suppose that we carry out the process described in the previous paragraph by selecting $\phi = \nu^T y$. As $\mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^\lambda(y)$, it is clear from Algorithm 2.2 that no pruning occurs during the last L iterations of Step 2 in Algorithm 2.2 applied to arguments $\text{TREE}^0(y), y, \lambda$, and $\mathcal{O}$. Therefore, the process from the previous paragraph returns the optimally pruned $\text{TREE}^\lambda(y)$. Thus, when $\mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^\lambda(y)$, we can simply plug in $\text{TREE}^\lambda(y)$, which has already been built, for $\text{TREE}(\mathcal{B},\nu,\lambda)$ in (2.21).

A.4.4 Proof of Proposition 2.4.4

We first show that we can express

$$\left\{\phi : g\left\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} \geq \lambda\right\} \tag{A.23}$$

as the solution set of a quadratic inequality in ϕ for $l = 0, \ldots, L-1$, where $g(\cdot)$ was defined in (2.3). Because only the numerator of $g\{R^{(l)}, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu)\}$ depends on ϕ, it will be useful to introduce the following concise notation:

$$h(R, \text{TREE}, y) \equiv \sum_{i \in R} (y_i - \bar{y}_R)^2 - \sum_{r \in \text{TERM}(R, \text{TREE})} \sum_{i \in r} (y_i - \bar{y}_r)^2. \tag{A.24}$$

We begin with the following lemma.

Lemma A.9 *Suppose that region R in* TREE *has children $R \cap \chi_{j,s,0}$ and $R \cap \chi_{j,s,1}$. Then, $h(R, \text{TREE}, y) = \text{GAIN}_R(y, j, s) + h(R \cap \chi_{j,s,1}, \text{TREE}, y) + h(R \cap \chi_{j,s,0}, \text{TREE}, y)$, where $\text{GAIN}_R(y, j, s)$ is defined in* (2.2).

A.4.4.0.1 Proof: The result follows from adding and subtracting $\sum_{i \in R \cap \chi_{j,s,1}} (y_i - \bar{y}_{R \cap \chi_{j,s,1}})^2$ and $\sum_{i \in R \cap \chi_{j,s,0}} (y_i - \bar{y}_{R \cap \chi_{j,s,0}})^2$ in (A.24) and noting that $\text{TERM}(R, \text{TREE}) = \text{TERM}(R \cap \chi_{j,s,1}, \text{TREE}) \cup \text{TERM}(R \cap \chi_{j,s,0})$. □

Recall that $\mathcal{B} = ((j_1, s_1, e_1), \ldots, (j_L, s_L, e_L))$ and that $\mathcal{R}(\mathcal{B}) = \{R^{(0)}, R^{(1)}, \ldots, R^{(L)}\}$. Lemma A.10 follows from Lemma A.9 and the fact that, due to the form of the vector ν, there are many regions R for which $h\{R, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu)\}$ does not depend on ϕ.

Lemma A.10 *For any $\tilde{\phi} \in S_{grow}(\mathcal{B}, \nu)$, we can decompose $h\{R^{(l)}, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu)\}$ as*

$$\begin{aligned} h\left\{R^{(l)}, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu)\right\} = & \sum_{l'=l+1}^{L} h\left\{R_{sib}^{(l')}, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\tilde{\phi}, \nu)\right\} \\ & + h\left\{R^{(L)}, \text{TREE}(\mathcal{B}, \nu, \lambda), y'(\tilde{\phi}, \nu)\right\} \\ & + \sum_{l'=l}^{L-1} \text{GAIN}_{R^{(l')}}\{y'(\phi, \nu), j_{l'+1}, s_{l'+1}\}, \end{aligned}$$

where $R_{sib}^{(l)}$ is the sibling of $R^{(l)}$ in $\text{TREE}(\mathcal{B}, \nu, \lambda)$.

A.4.4.0.2 Proof: Repeatedly applying Lemma A.9 yields

$$h\left\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} = \sum_{l'=l+1}^{L} h\left\{R_{sib}^{(l')}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} + h\left\{R^{(L)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} + \sum_{l'=l}^{L-1} \text{GAIN}_{R^{(l')}}\{y'(\phi,\nu), j_{l'+1}, s_{l'+1}\}. \quad \text{(A.25)}$$

For $l' = l+1,\ldots,L-1$, $h\left\{R_{sib}^{(l')}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} = h\left\{R_{sib}^{(l')}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\tilde{\phi},\nu)\right\}$ because $R_{sib}^{(l')}$ only contains observations where $[y'(\phi,\nu)]_i = [y'(\tilde{\phi},\nu)]_i$. Similarly, $h\left\{R^{(L)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} = h\left\{R^{(L)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\tilde{\phi},\nu)\right\}$ and $h\left\{R_{sib}^{(L)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} = h\left\{R_{sib}^{(L)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\tilde{\phi},\nu)\right\}$ because for $i \in R^{(L)}$ and $i \in R_{sib}^{(L)}$, $\{y'(\phi,\nu)\}_i$ and $\{y'(\tilde{\phi},\nu)\}_i$ only differ by a constant shift. Plugging these two facts into (A.25) completes the proof. □

We can now write (A.23) as

$$\begin{aligned}
&\left\{\phi : g\left\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} \geq \lambda\right\} \\
&= \left\{\phi : h\left\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} \geq \lambda\left[|\,\text{TERM}\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda)\}| - 1\right]\right\} \\
&= \left\{\phi : \sum_{l'=l}^{L-1} \text{GAIN}_{R^{(l')}}\{y'(\phi,\nu), j_{l'+1}, s_{l'+1}\} \geq \lambda\left[|\,\text{TERM}\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda)\}| - 1\right] - \right. \\
&\qquad \left. \sum_{l'=l+1}^{L} h\left\{R_{sib}^{(l')}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\tilde{\phi},\nu)\right\} - h\left\{R^{(L)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\tilde{\phi},\nu)\right\}\right\} \\
&= \left\{\phi : \sum_{l'=l}^{L-1}\left[a\left\{R^{(l')}, j_{l'+1}, s_{l'+1}\right\}\phi^2 + b\left\{R^{(l')}, j_{l'+1}, s_{l'+1}\right\}\phi + c\left\{R^{(l')}, j_{l'+1}, s_{l'+1}\right\}\right] \geq \gamma_l\right\}, \quad \text{(A.26)}
\end{aligned}$$

where the functions $a(\cdot), b(\cdot)$, and $c(\cdot)$ were defined in (A.12) in Appendix A.4.2, and where

$$\gamma_l \equiv \lambda\left[|\,\text{TERM}\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda)\}| - 1\right] - \sum_{l'=l+1}^{L} h\left\{R_{sib}^{(l')}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\tilde{\phi},\nu)\right\} - h\left\{R^{(L)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\tilde{\phi},\nu)\right\}$$

is a constant that does not depend on ϕ. The first equality simply applies the definitions of $h(\cdot)$ and $g(\cdot)$. The second equality follows from Lemma A.10 and moving terms that do not

depend on ϕ to the right-hand-side. The third equality follows from plugging in notation from Appendix A.4.2 and defining the constant γ_l for convenience. Thus, (A.23) is quadratic inequality in ϕ. We now just need to argue that its coefficients can be obtained efficiently.

We need to compute the coefficients in (A.26) for $l = 0, \ldots, L-1$. The quantities $a\{R^{(l')}, j_{l'}, s_{l'+1}\}$, $b\{R^{(l')}, j_{l'+1}, s_{l'+1}\}$, and $c\{R^{(l')}, j_{l'+1}, s_{l'+1}\}$ for $l' = 0, \ldots, L-1$ were already computed while computing $S_{grow}(\mathcal{B}, \nu)$. To get the coefficients for the left hand side of (A.26) for each $l = 0, \ldots, L-1$, we simply need to compute L partial sums of these quantities, which takes $O(L)$ operations. As we are assuming that we have access to $\textsc{tree}(\mathcal{B}, \nu, \lambda)$, computing $h\{R, \textsc{tree}(\mathcal{B}, \nu, \lambda), y'(\tilde{\phi}, \nu)\}$ requires $O(n)$ operations. Therefore, computing γ_0 takes $O(nL)$ operations. By storing partial sums during the computation of γ_0, we can subsequently obtain γ_l for $l = 1, \ldots, L-1$ in constant time.

We have now seen that we can obtain the coefficients needed to express (A.23) as a quadratic function of ϕ for $l = 0, \ldots, L-1$ in $O(nL)$ total operations. Once we have these quantities, we can compute each set of the form (A.23) in constant time using the quadratic equation.

It remains to compute

$$S^{\lambda}(\mathcal{B}, \nu) = S_{grow}(\mathcal{B}, \nu) \bigcap \left[\bigcap_{l=0}^{L-1} \left\{ \phi : g\left\{ R^{(l)}, \textsc{tree}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu) \right\} \geq \lambda \right\} \right]. \tag{A.27}$$

Recall from Proposition 2.4.1 that $S_{grow}(\mathcal{B}, \nu)$ is the intersection of $O(npL)$ quadratic sets. Thus, in the worst case, $S_{grow}(\mathcal{B}, \nu)$ has $O(npL)$ disjoint components, and so this final intersection involves $O(npL)$ components. Thus, we can compute $S^{\lambda}(\mathcal{B}, \nu)$ in $O\{npL \times log(npL)\}$ operations [Bourgon, 2009].

The following lemma explains why, similar to Proposition 2.4.2, computation time can be reduced in the special case where $\nu = \nu_{sib}$ and $\mathcal{B} = \textsc{branch}\{R_A, \textsc{tree}^{\lambda}(y)\}$.

Lemma A.11 *When* $\mathcal{B} = \textsc{branch}\{R_A, \textsc{tree}^{\lambda}(y)\}$, *the set* $\{\phi : g\{R^{(l)}, \textsc{tree}(\mathcal{B}, \nu, \lambda), y'(\phi, \nu_{sib})\} \geq \lambda\}$ *has the form* $(-\infty, a_l) \cup (b_l, \infty)$, *where* $a_l \leq 0 \leq b_l$.

Therefore, we can compute $\bigcap_{l=0}^{L-1}\left\{\phi : g\left\{R^{(l)}, \text{TREE}(\mathcal{B},\nu,\lambda), y'(\phi,\nu)\right\} \geq \lambda\right\}$ *as*

$$\bigcap_{l=0}^{L-1}(-\infty, a_l) \cup (b_l, \infty) = (-\infty, \min_{0\leq l\leq L-1} a_l) \cup (\max_{0\leq l\leq L-1} b_l, \infty)$$

in $O(L)$ *operations. Furthermore, we can compute* (A.27) *in constant time.*

A.4.4.0.3 Proof: As $\mathcal{R}(\mathcal{B}) \subseteq \text{TREE}^{\lambda}(y)$, we can let $\tilde{\phi} \in S_{grow}(\mathcal{B},\nu)$ from Lemma A.10 be $\nu^T y$ such that $y'(\tilde{\phi},\nu) = y$. We can then apply Lemma A.3 to note that

$$\sum_{l'=l}^{L-1} \text{GAIN}_{R^{(l')}}\{y'(\phi,\nu_{sib}), j_{l'+1}, s_{l'+1}\} = \left[\sum_{l'=l}^{L-2} \text{GAIN}_{R^{(l')}}\{y, j_{l'+1}, s_{l'+1}\}\right] + \text{GAIN}_{R^{(L-1)}}\{y'(\phi,\nu_{sib}), j_L, s_L\}.$$

Thus, when $\nu = \nu_{sib}$ and $\mathcal{B} = \text{BRANCH}\{R_A, \text{TREE}^{\lambda}(y)\}$, we can rewrite (A.26) with all of the terms corresponding to $\text{GAIN}_{R^{(l')}}\{y'(\phi,\nu_{sib}), j_{l'+1}, s_{l'+1}\}$ for $l' = l+1,\ldots,L-2$ moved into the constant on the right-hand-side. This lets us rewrite (A.26) as $\{\phi : Gain_{R^{(L-1)}}\{y'(\phi,\nu_{sib}), j_L, s_L\} \geq \tilde{\gamma}_l\}$, where $\tilde{\gamma}_l$ is an updated constant that does not depend on ϕ.

To prove that $\{\phi : Gain_{R^{(L-1)}}\{j_L, s_L, y'(\phi,\nu_{sib})\} \geq \tilde{\gamma}_l\}$ has the form $(-\infty, a_l) \cup (b_l, \infty)$ for $a_l \leq 0 \leq b_l$, first recall from Lemma A.4 that $Gain_{R^{(L-1)}}\{j_L, s_L, y'(\phi,\nu_{sib})\}$ is a quadratic function of ϕ. It then suffices to show that this quadratic has a non-negative second derivative and achieves its minimum when $\phi = 0$. The second derivative of this quadratic is $a\left\{R^{(L-1)}, j_L, s_L\right\} = ||\nu_{sib}||_2^{-4}\nu_{sib}^T M_{R^{(L-1)},j_L,s_L}\nu_{sib}$, which is non-negative by Lemma A.4. From Lemma A.4, $Gain_{R^{(L-1)}}\{j_L, s_L, y'(\phi,\nu_{sib})\}$ is non-negative. It equals 0 when $\phi = 0$, because when $\phi = 0$ then $\bar{y}_{R^{(L-1)}\cap\chi_{j_L,s_L,1}} = \bar{y}_{R^{(L-1)}\cap\chi_{j_L,s_L,0}}$.

Intersecting L sets of the form $(-\infty, a_l) \cup (b_l, \infty)$ for $a_l \leq 0 \leq b_l$ only takes $O(L)$ operations, because we simply need to identify the minimum a_l and maximum b_l. Finally, Lemma A.7 ensures that $S_{grow}(\mathcal{B},\nu_{sib})$ has at most two disjoint intervals, and so the final intersection with $S_{grow}(\mathcal{B},\nu_{sib})$ takes only $O(1)$ operations. □

A.4.5 Proof of Proposition 2.4.5

Let $\mathcal{B} = \text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}$, let $\mathcal{R}(\mathcal{B}) = \{R^{(0)}, \ldots, R^{(L)}\}$, and let $\nu = \nu_{sib}$ (2.6). Applying the expression given for $\{y'(\phi, \nu_{sib})\}_i$ in Section 2.3.3 (which follows from algebra), we immediately see that Condition 2.1 holds with $R_A = R^{(L)}$ and $R_B = R^{(L-1)} \cap \chi_{j_L, s_L, 1-e_L}$.

Let $\mathcal{B} = \pi\left[\text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}\right]$ and let $\nu = \nu_{reg}$ (2.14). Note that $\pi\left[\text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}\right]$ induces the same region $R^{(L)}$ as the unpermuted $\text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}$. Regardless of the permutation, the induced $R^{(L)}$ is equal to R_A. Applying the expression for $\{y'(\phi, \nu_{reg})\}_i$ given in Section 2.3.3, Condition 2.1 holds with constant $c_2 = 0$.

A.5 Proofs for Section 2.4.3

A.5.1 Proof of Proposition 2.4.6

As stated in Proposition 2.4.6, let

$$p^Q_{reg}(y) = pr_{H_0}\left\{|\nu_{reg}^{\mathrm{T}} Y - c| \geq |\nu_{reg}^{\mathrm{T}} y - c| \mid \bigcup_{\pi \in Q} \mathcal{R}(\pi[\text{BRANCH}\{R_A, \text{TREE}^\lambda(y)\}]) \subseteq \text{TREE}^\lambda(Y), \mathcal{P}^\perp_{\nu_{reg}} Y = \mathcal{P}^\perp_{\nu_{reg}} y\right\}.$$

First, we will show that the test based on $p^Q_{reg}(y)$ controls the selective Type 1 error rate, defined in (2.4); this is a special case of Proposition 3 from Fithian et al. [2014].

Define $\mathcal{E}_1 = \{Y : R_A \in \text{TREE}^\lambda(Y)\}$, $\mathcal{E}_2 = \{Y : \mathcal{P}^\perp_{\nu_{reg}} Y = \mathcal{P}^\perp_{\nu_{reg}} y\}$, and $\mathcal{E}_3 = \left\{Y : \bigcup_{\pi \in Q} \mathcal{R}\left(\pi\left[\text{BRANCH}\left\{R_A, \text{TREE}^\lambda(y)\right\}\right]\right) \subseteq \text{TREE}^\lambda(Y)\right\}$. Recall that the test of $H_0 : \nu_{reg}^{\mathrm{T}} \mu = c$ based on $p^Q_{reg}(y)$ controls the selective Type 1 error rate if, for all $\alpha \in [0, 1]$, $pr_{H_0}\{p^Q_{reg}(Y) \leq \alpha \mid \mathcal{E}_1\} \leq \alpha$. By construction,

$$pr_{H_0}\left\{p^Q_{reg}(Y) \leq \alpha \mid \mathcal{E}_2 \cap \mathcal{E}_3\right\} = E\left[1_{\{pr_{H_0}\left(|\nu_{reg}^{\mathrm{T}} Y - c| \geq |\nu_{reg}^{\mathrm{T}} y - c| \mid \mathcal{E}_2 \cap \mathcal{E}_3\right) \leq \alpha\}} \mid \mathcal{E}_2 \cap \mathcal{E}_3\right] = \alpha.$$

Let $\psi^Q_{R_A} = 1_{\{pr_{H_0}\left(|\nu_{reg}^{\mathrm{T}} Y - c| \geq |\nu_{reg}^{\mathrm{T}} y - c| \mid \mathcal{E}_2 \cap \mathcal{E}_3\right) \leq \alpha\}}$. An argument similar to that of Lemma 2.2

indicates that $\mathcal{E}_3 \subseteq \mathcal{E}_1$. The law of total expectation then yields

$$E(\psi_{R_A}^Q \mid \mathcal{E}_1) = E\left\{E(\psi_{R_A}^Q \mid \mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3) \mid \mathcal{E}_1\right\} = E\left\{E(\psi_{R_A}^Q \mid \mathcal{E}_2 \cap \mathcal{E}_3) \mid \mathcal{E}_1\right\} = E(\alpha \mid \mathcal{E}_1) = \alpha.$$

Thus, the test based on $p_{reg}^Q(y)$ controls the selective Type 1 error rate. We omit the proof that $p_{reg}^Q(y)$ can be computed as

$$p_{reg}^Q(y) = pr_{H_0}\left\{|\phi - c| \geq |\nu_{reg}^{\mathrm{T}} y - c| \mid \phi \in \bigcup_{\pi \in Q} S^\lambda\left(\pi[\textsc{branch}\{R_A, \textsc{tree}^\lambda(y)\}], \nu_{reg}\right)\right\}$$

for $\phi \sim N(c, \|\nu_{reg}\|_2^2 \sigma^2)$, as the proof is similar to the proof of Theorem 2.1.

A.6 Effect of using the computationally efficient alternative in Section 2.4.3

In this section, we investigate the effect of using $p_{reg}^{\mathcal{I}}(y)$ from (2.22) rather than $p_{reg}(y)$ from (2.15) on power. We also investigate the effect of using (2.23) rather than (2.17) on the width of confidence intervals for $\nu_{reg}^{\mathrm{T}}\mu$.

We generate data as described in Section 2.5.1, but for simplicity we restrict our attention to the case where $a = 1$ (corresponding to the center panel of Figure 2.3).

For each tree that we build, we consider (a) testing $H_0 : \nu_{reg}^{\mathrm{T}}\mu = 0$ and (b) constructing a confidence interval for $\nu_{reg}^{\mathrm{T}}\mu$, for each region appearing at the third level of the tree. For each test, we compare the test that uses the full conditioning set (i.e. that uses $p_{reg}(y)$ from (2.15)) to the test that uses the identity permutation only (i.e. that uses $p_{reg}^{\mathcal{I}}(y)$ from (2.22)). For each interval, we compare the method that uses the full conditioning set (i.e. (2.17)) to the method that uses the identity permutation only (i.e. (2.23)). The results are displayed in Figure A.2.

The left panel of Figure A.2 shows that the power loss resulting from using (2.22) instead of (2.15) is negligible. In fact, we need to zoom in on the left panel, as shown in the center panel, to see any separation between the power curves. We see in the center panel that power is lower when (2.22) is used, though we emphasize that the differences in power are

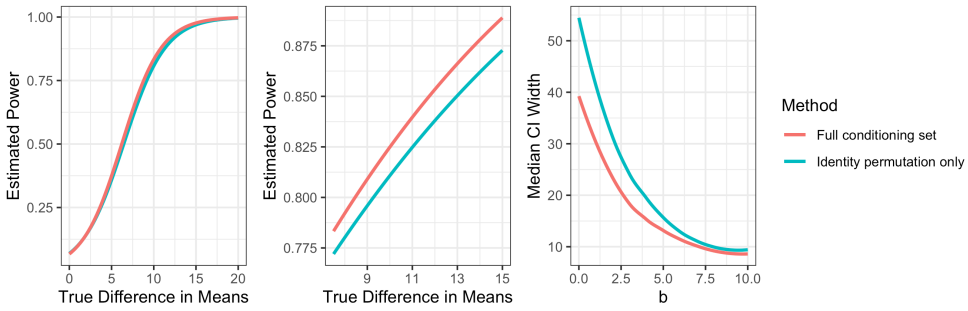


Figure A.2: Simulation results comparing inference based on the full conditioning set to inference based on the identity permutation only (see Section 2.4.3). The left panel shows power curves. The center panel zooms in on one section of the left panel. The right panel shows median widths of confidence intervals.

extremely small.

We see in the right panel of Figure A.2 that the computationally efficient conditioning set has a more noticeable impact on the median width of our confidence intervals. As expected, the confidence intervals are narrower when we use the full conditioning set (i.e. (2.17)). However, this difference is most noticeable when b is small. When b is small, in which case the confidence intervals are wide even when the full conditioning set is used. Thus, the amount of precision lost overall by constructing confidence intervals using the identity permutation only (i.e. (2.23)) is not of practical importance.

Based on these results, we recommend using the identity permutation in practice, because it is the most computationally efficient choice *and* does not meaningfully reduce power or precision compared to the full conditioning set.

A.7 Robustness to non-normality

In this section, we explore the performance of the selective Z-test under a global null when the normality assumption on Y is violated. For four choices of cumulative distribution function F, we generate $Y_i \overset{\text{i.i.d.}}{\sim} F$ such that all observations have the same expected value. We set $n = 200, p = 10$, and we grow trees to a maximum depth of 3. We plug in $(n-1)^{-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$

as an estimate of σ^2, as it does not make sense to assume known variance for distributions with a mean-variance relationship. Figure A.3 displays quantile-quantile plots of the p-values for testing $H_0 : \nu_{sib}^{\mathrm{T}}\mu = 0$, using the test in (2.8).

Figure A.3 shows that despite the fact that our proposed selective inference framework was derived under a normality assumption, it yields approximately uniformly distributed p-values for Poisson(10), Bernoulli(0.5), and Gamma(1,10) data. We suspect that this is because $\mathcal{P}_{\nu}^{\perp}Y$ is approximately independent of $\nu^{\mathrm{T}}Y$ for these distributions (see the proof of Theorem 2.1 in Appendix A.2). In the case of Bernoulli(0.1) data, which represents a particularly extreme violation of the normality assumption, the p-values from our selective Z-test are not uniformly distributed. As mentioned in Section 2.7, future work could involve characterizing conditions for F under which our selective Z-tests will approximately control the selective Type 1 error.

A.8 Alternate Box Lunch Study analysis

Figure A.4 is the same as the left panel of Figure 2.9, but the selective Z-inference is carried out with $\hat{\sigma}_{\mathrm{cons}}$, from Section 2.5.7, rather than $\hat{\sigma}_{\mathrm{SSE}}$. The takeaways presented in Section 2.6 do not change.

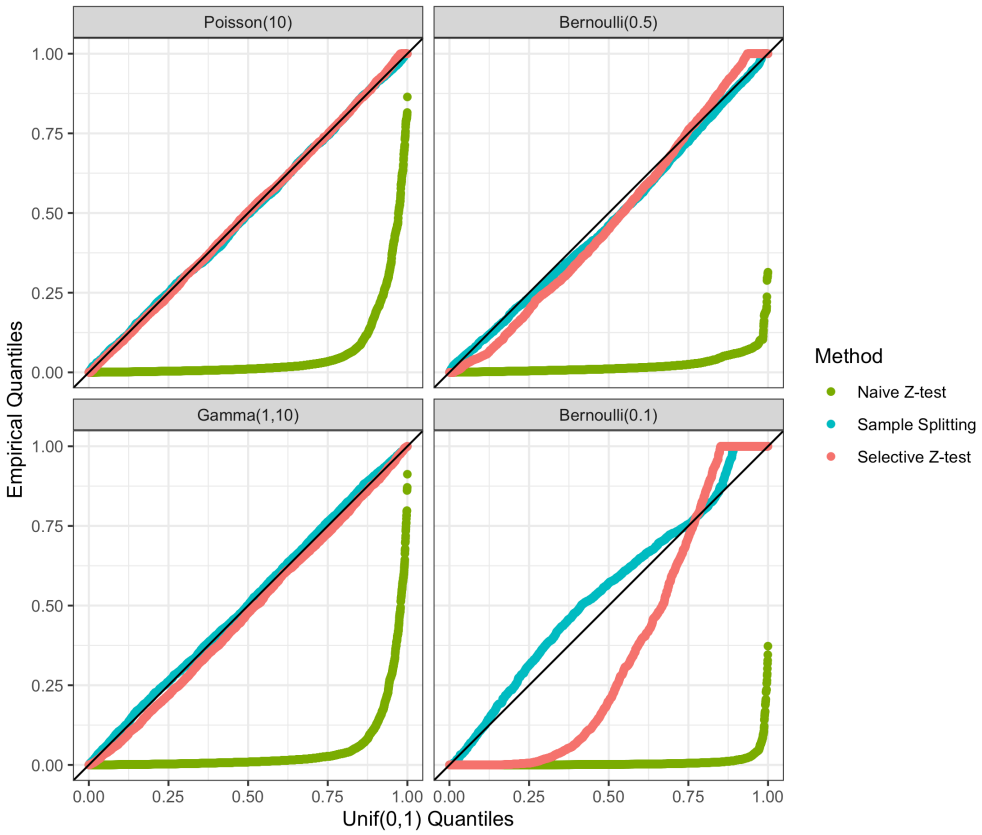


Figure A.3: Quantile-quantile plots of the p-values for testing $H_0 : \nu_{sib}^{\mathrm{T}}\mu = 0$ under a global null. A naive Z-test (green), sample splitting (blue), and selective Z-test (pink) were performed; see Section 2.5.2.

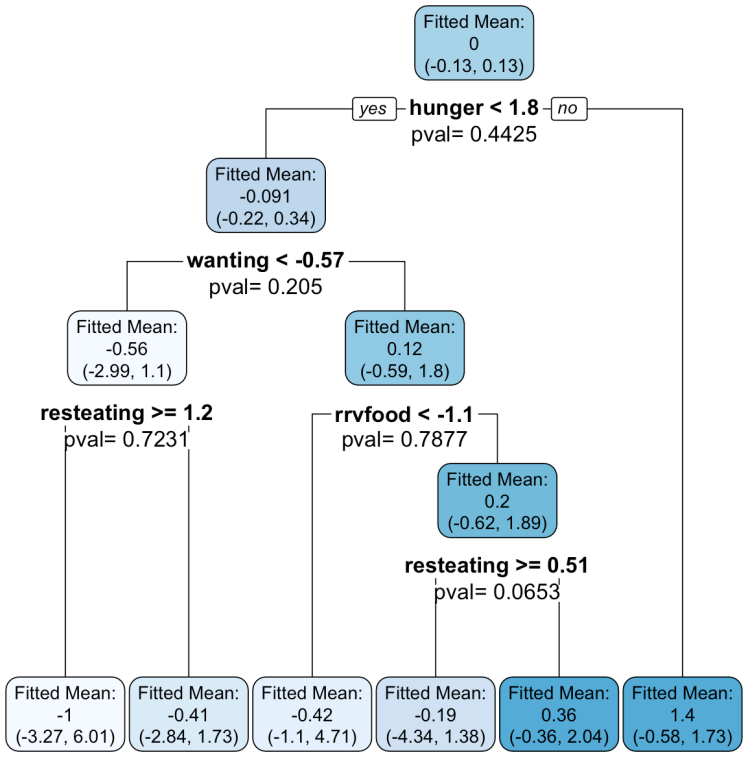


Figure A.4: A CART tree fit to the Box Lunch Study data. Each split has been labeled with a p-value (2.8), and each region has been labeled with a confidence interval (2.23). Inference is carried out by plugging in $\hat{\sigma}_{\text{cons}}$, from Section 2.5.7, as an estimate of σ.

Appendix B

APPENDICES FOR CHAPTER 3

B.1 Illustrating the issues with the standard Seurat and Monocle3 pipelines

In this appendix, we elaborate on our claim that the popular `R` packages `Seurat` [Hoffman et al., 2021] and `Monocle3` [Pliner et al., 2020] promote use of the problematic two step procedure that we introduced in Section 3.1 and that we refer to as "double dipping". The individual functions in both packages are sound. However, because the pipelines suggested in the package vignettes make use of the same data for both latent variable estimation and inference ("double dipping"), they lead to artificially small p-values that fail to control the Type 1 error rate.

The `Seurat` "guided clustering tutorial" [Hoffman et al., 2022] suggests that, after applying numerous preprocessing steps, the function `FindClusters()` should be applied to the data to estimate clusters, and then subsequently the function `FindMarkers()` should be applied to the same data to test for differential expression across these estimated clusters. The `FindMarkers()` function returns p-values from standard statistical tests (Wilcoxon rank sum tests by default, or other tests if different arguments are supplied) that do not account for the fact that the clusters were estimated from the data.

Similarly, the `Monocle3` tutorial contains a section called "finding genes that change as a function of pseudotime" [Pliner et al., 2022]. After preprocessing and dimension reduction, the functions `learn_graph()` and `order_cells()` are applied to the data to estimate pseudotime. Subsequently, the function `graph_test()` is applied to the same data. This function applies a standard Moran's I test to test for spatial dependence of gene expression across pseudotime space for each gene, which does not correct for the fact that the pseudotime space was estimated from the data.

To demonstrate the deficiencies with the above pipelines, we perform a simple simulation study. We generate 500 datasets $X \in \mathbb{Z}_{\geq 0}^{500 \times 200}$ where $\mathbf{X}_{ij} \sim \text{Poisson}(5)$. Under this data generating mechanism, all cells are drawn from the same distribution and so no genes are differentially expressed across any latent variables. We then carry out the following two methods for each dataset X.

- **Seurat (double dipping, as is done in the Seurat tutorial):** We convert X into a `Seurat` object. We run the required preprocessing steps, and then we apply the `FindClusters()` function with resolution = 1 to the object. We then run `FindMarkers()` on the object to test for differential expression between the first and second estimated clusters. We save the Wilcoxon rank-sum p-values for each gene returned by the `FindMarkers()` function.

- **Seurat (count splitting):** We first perform count splitting on the data to obtain X^{train} and X^{test}. We create a Seurat object containing the counts X^{train}, and preprocess X^{train} and apply `FindClusters()` with the same arguments as above. We then add the X^{test} counts to the `Seurat` object as an additional assay, and we run `FindMarkers()` with the same arguments as above but with the additional argument `assay="test"`. We save the p-values for each gene returned by the `FindMarkers()` function.

- **Monocle3 (double dipping, as in the Monocle3 tutorial):** We first convert the data into a `cell_data_set` object. We then run the required preprocessing steps, and then run the functions `learn_graph()` and `order_cells()` to estimate pseudotime. We run all functions with their default settings, and for `order_cells()` we set the root cell to be the first cell in the dataset. Finally, we run `graph_test()` and save the Wilcoxon rank-sum p-values returned for each gene.

- **Monocle3 (count splitting):** We first perform count splitting on the data to obtain X^{train} and X^{test}. We create a `cell_data_set` object containing X^{train}. We preprocess X^{train} and then run `learn_graph()` and `order_cells()` on the training object, with

the same arguments as above. We then create a new `cell_data_set` object that is a copy of the training set object, and thus stores the same graph and pseudotime information. However, we update the `counts` attribute of this dataset to store X^{test}. We then run `graph_test()` on this new object and save the p-value returned for each gene.

Figure B.1 shows uniform QQ plots obtained from each of the four methods above. For both `Monocle3` and `Seurat`, the pipeline that uses the same data for clustering or pseudotime estimation and differential expression analysis leads to p-values that are too small, while the count splitting pipeline yields p-values that are uniformly distributed. This demonstrates that estimating a latent variable and testing for differential expression on the same data leads to invalid p-values.

Code to reproduce Figure B.1 is available on our `R` package website at `anna-neufeld.github.io/countsplit`.

B.2 Connections to Batson et al. [2019]

To evaluate the goodness-of-fit of low-rank approximations of scRNA-seq expression matrices, with the ultimate goal of selecting optimal hyperparameters (such as the number of principal components to keep), Batson et al. [2019] seek to obtain independent training and test sets. They propose *molecular cross validation* (MCV), a method for obtaining such sets under the assumption that $X_{ij} \sim \text{Binomial}(\Omega_{ij}, p_i)$, where Ω_{ij} is the true number of mRNA molecules for gene j in cell i and p_i is the probability that a molecule in cell i is observed. The splitting method involves three steps:

1. $X_{ij}^{\text{train}} \sim \text{Binomial}\left(X_{ij}, \epsilon\right)$
2. $X_{ij}^{\text{both}} \sim \text{Binomial}\left(X_{ij}^{\text{train}}, p_i''\right)$
3. $X_{ij}^{\text{test}} = X_{ij} - X_{ij}^{\text{train}} + X_{ij}^{\text{both}}$.

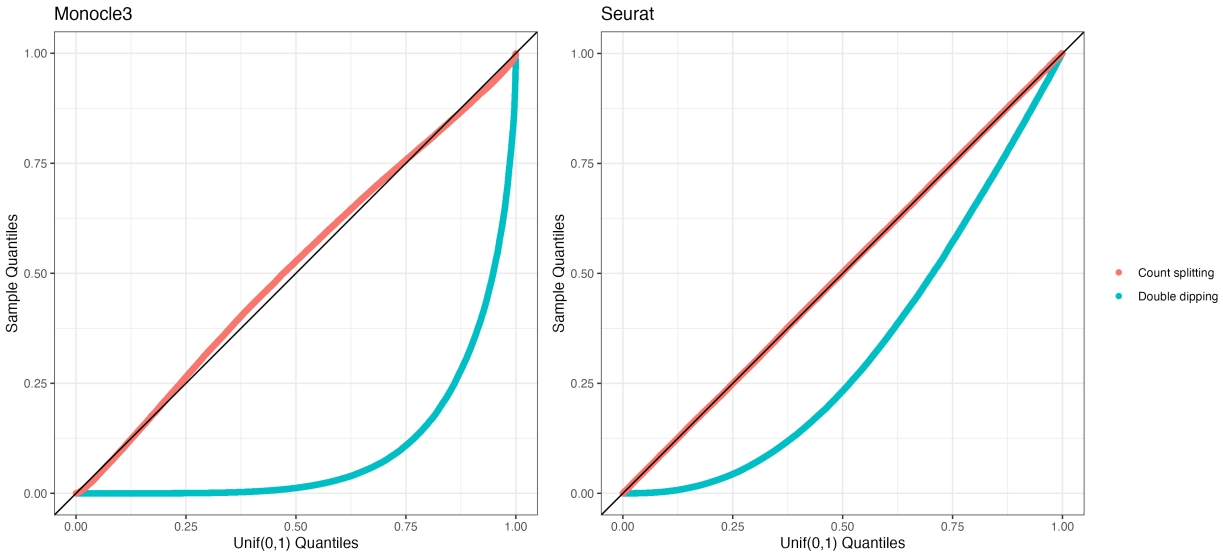


Figure B.1: Uniform QQ-plots of p-values under a global null, obtained using four different methods.

Under the binomial assumption, if p_i'' is chosen appropriately, then X_{ij}^{train} and X_{ij}^{test} are independent. Unfortunately, choosing the appropriate p_i'' requires knowledge of the parameter p_i.

As p_i is typically unknown in practice, the authors suggest a simplification of the three-step process above in which X_{ij}^{both} is taken to be 0. Under this simplification, MCV becomes identical to count splitting (Step 0 of Algorithm 3.1 in the main text).

The authors' explanation for setting $X_{ij}^{\text{both}} = 0$ is as follows. They imagine that X_{ij} counts the total number of unique molecules seen through two separate, shallower sequencing experiments (which gave rise to X_{ij}^{train} and X_{ij}^{test}). In this framework, X_{ij}^{both} denotes the molecules that were counted by both experiments and thus need to be subtracted out. The authors note that, since scRNA-seq experiments tend to be quite shallow (i.e. the values of p_i are small), the overlap between two even shallower experiments is likely to be negligible. We provide an alternate justification: when the p_i are small and the Ω_{ij} are large, then the Poisson approximation to the binomial distribution holds, and so independence is given by Proposition 3.4.1.

Thus, while developed for a different goal and under different assumptions, the procedure of Batson et al. [2019] provides some justification for count splitting under a binomial assumption.

B.3 Simulation comparing count splitting to selective inference

Gao et al. [2022] provide a selective inference approach for testing the null hypothesis that two clusters estimated via hierarchical clustering have the same mean vector, under an assumption of multivariate normality. More specifically, they assume

$$\mathbf{X} \sim \mathcal{MN}_{n\times q}\left(\mu, \mathbf{I}_n, \sigma^2\mathbf{I}_q\right), \tag{B.1}$$

and seek to test

$$H_0 : \bar{\mu}_{\hat{\mathcal{C}}_1} = \bar{\mu}_{\hat{\mathcal{C}}_2}, \tag{B.2}$$

where $\hat{\mathcal{C}}_1$ and $\hat{\mathcal{C}}_2$ index the observations assigned to the two clusters and where, for any $G \subseteq \{1, \ldots, n\}$, we define $\bar{\mu}_G = \frac{1}{|i \in G|} \sum_{i \in G} \mu_i$. A naive Wald test for this null hypothesis does not control the Type 1 error rate because it ignores the fact that $\hat{\mathcal{C}}_1$ and $\hat{\mathcal{C}}_2$ are functions of the data X. Gao et al. [2022] propose a selective Z-test that conditions on, among other things, the event that $\hat{\mathcal{C}}_1$ and $\hat{\mathcal{C}}_2$ were output by the hierarchical clustering algorithm. As shown in their paper, this method controls the Type 1 error rate when the data truly come from a multivariate normal distribution.

While not the focus of this paper, count splitting can also test the null hypothesis in (B.2) using a modification of Algorithm 3.1 from the main text.

To compare count splitting to selective inference in terms of Type 1 error rate control in this setting, we generate 1000 datasets where $n = 200, p = 10$ and $X_{ij} \sim \text{Poisson}(5)$. For each dataset, we do the following:

Double dipping: Run hierarchical clustering with average linkage on $\log(X + 1)$ to obtain two clusters $\hat{\mathcal{C}}_1$ and $\hat{\mathcal{C}}_2$. Use the naive test that double dips described in Gao et al. [2022] on $\log(X + 1)$ to test $H_0 : \bar{\mu}_{\hat{\mathcal{C}}_1} = \bar{\mu}_{\hat{\mathcal{C}}_2}$. For this Wald test, estimate σ using the "conservative estimate" from Gao et al. [2022].

Selective: Run hierarchical clustering with average linkage on $\log(X + 1)$ to obtain two clusters $\hat{\mathcal{C}}_1$ and $\hat{\mathcal{C}}_2$. Use the selective test of Gao et al. [2022] to test $H_0 : \bar{\mu}_{\hat{\mathcal{C}}_1} = \bar{\mu}_{\hat{\mathcal{C}}_2}$. Estimate σ using the "conservative estimate" from **?**.

Count Split: Run Step 0 (count splitting) of Algorithm 3.1 with $\epsilon = 0.5$ to obtain X^{train} and X^{test}. Run hierarchical clustering with average linkage on $\log(X^{\text{train}} + 1)$ to assign each cell to $\hat{\mathcal{C}}_1$ or $\hat{\mathcal{C}}_2$. Use the Wald test described in Gao et al. [2022] on $\log(X^{\text{test}} + 1)$ to test $H_0 : \bar{\mu}_{\hat{\mathcal{C}}_1} = \bar{\mu}_{\hat{\mathcal{C}}_2}$. For this Wald test, estimate σ using the "conservative estimate" from Gao et al. [2022], computed on the test set.

The results of this experiment are shown in Figure B.3. While selective inference improves upon the naive method that double dips, Figure B.2 shows that it fails to control the Type 1

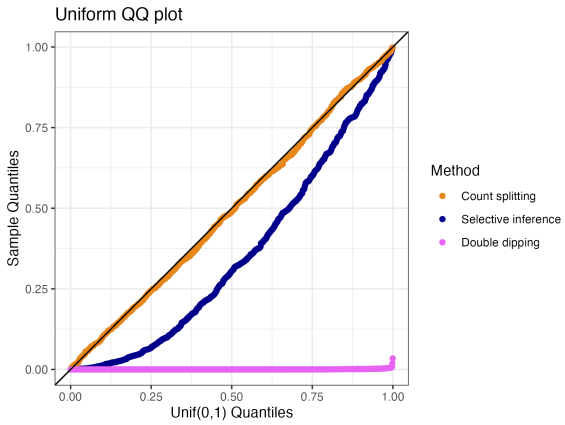


Figure B.2: Uniform QQ-plot of p-values under a global null. Count splitting controls the Type 1 error while selective inference does not.

error rate for log-transformed Poisson data. This illustrates why the dependence of selective inference on a normality assumption makes it inadequate for scRNA-seq applications.

B.4 Implementation details for Figure 3.2(a)

In Figure 3.2(a), the double dipping method, cell splitting, and count splitting are all carried out as specified in Sections 3.3 and 3.4. The p-values reported are the default p-values returned by the `glm` function in `R` (Wald p-values).

The `jackstraw` `R` package [Chung et al., 2021] does not allow for fitting GLMs or for arbitrary latent variable estimation techniques. To obtain the jackstraw results for Figure 3.2(a), we implement the algorithm described in Section 3.3.6, which is similar to the proposal of Chung and Storey [2015] but allows for a GLM and an arbitrary latent variable estimation technique. We set $b = 100$ and $s = 10$ for our implementation.

The `pseudotimeDE` `R` package [Dongyuan Song and Nguyen, 2021] does allow for an arbitrary pseudotime estimation technique, but it fits negative binomial generalized additive models (GAMs) by default, and conducts likelihood ratio tests rather than Wald tests. To remain true to Song and Li [2021], the p-values for pseudotimeDE that are shown in Figure 3.2(a) are the default p-values returned by their `R` package (i.e. from GAMs and not

GLMs). We estimate pseudotime using the first principal component of the log transformed matrix (as for the other methods). We then pass this estimate of pseudotime into the `R` package's `runPseudotimeDE()` function with its default settings. To compute each empirical p-value, we use $B = 100$ subsets of the data, each containing 80% of the observations.

B.5 Proof of Propositions from Section 3.4

B.5.1 Proof of Proposition 3.4.2

Note that $\mathbf{X}_{ij}^{\text{train}} \mid \{\mathbf{X}_{ij} = X_{ij}\} \sim \text{Binomial}(X_{ij}, \epsilon)$ and $\mathbf{X}_{ij}^{\text{test}} \mid \{\mathbf{X}_{ij} = X_{ij}\} \sim \text{Binomial}(X_{ij}, 1-\epsilon)$. The first statement is by construction and the second follows from swapping the roles of successes and failures in the binomial experiment. We now derive the marginal variance of each distribution from the known conditional mean and variance:

$$\begin{aligned}
\text{Var}(\mathbf{X}_{ij}^{\text{test}}) &= \text{E}\left[\text{Var}\left[\mathbf{X}_{ij}^{\text{test}} \mid \mathbf{X}_{ij}\right]\right] + \text{Var}\left[\text{E}\left[\mathbf{X}_{ij}^{\text{test}} \mid \mathbf{X}_{ij}\right]\right] \\
&= \text{E}\left[(1-\epsilon)\epsilon\mathbf{X}_{ij}\right] + \text{Var}\left[(1-\epsilon)\mathbf{X}_{ij}\right] \\
&= (1-\epsilon)\epsilon\Lambda_{ij} + (1-\epsilon)^2\left(\frac{\Lambda_{ij}^2}{b_j} + \Lambda_{ij}\right) \\
&= (1-\epsilon)\Lambda_{ij} + \frac{(1-\epsilon)^2\Lambda_{ij}^2}{b_j}, \\
\text{Var}(\mathbf{X}_{ij}^{\text{train}}) &= \text{E}\left[\text{Var}\left[\mathbf{X}_{ij}^{\text{train}} \mid \mathbf{X}_{ij}\right]\right] + \text{Var}\left[\text{E}\left[\mathbf{X}_{ij}^{\text{train}} \mid \mathbf{X}_{ij}\right]\right] \\
&= (1-\epsilon)\epsilon\Lambda_{ij} + \epsilon^2\left(\frac{\Lambda_{ij}^2}{b_j} + \Lambda_{ij}\right) \\
&= \epsilon\Lambda_{ij} + \frac{\epsilon^2\Lambda_{ij}^2}{b_j}.
\end{aligned}$$

Next note that

$$\text{Var}(\mathbf{X}_{ij}^{\text{train}}) + \text{Var}(\mathbf{X}_{ij}^{\text{test}}) = \Lambda_{ij} + \frac{(\epsilon^2 + (1-\epsilon)^2)\Lambda_{ij}^2}{b_j},$$

and that since $\mathbf{X}_{ij} = \mathbf{X}_{ij}^{\text{train}} + \mathbf{X}_{ij}^{\text{test}}$,

$$\begin{aligned}
\text{Cov}\left(\mathbf{X}_{ij}^{\text{train}}, \mathbf{X}_{ij}^{\text{test}}\right) &= \frac{1}{2}\left(\text{Var}\left(\mathbf{X}_{ij}\right) - \text{Var}\left(\mathbf{X}_{ij}^{\text{train}}\right) - \text{Var}\left(\mathbf{X}_{ij}^{\text{test}}\right)\right) \\
&= \frac{1}{2}\left(\Lambda_{ij} + \frac{\Lambda_{ij}^2}{b_j} - \Lambda_{ij} - \frac{(\epsilon^2 + (1-\epsilon)^2)\Lambda_{ij}^2}{b_j}\right) \\
&= \frac{1}{2}\left(\frac{2(\epsilon(1-\epsilon))\Lambda_{ij}^2}{b_j}\right) = \frac{\epsilon(1-\epsilon)\Lambda_{ij}^2}{b_j}.
\end{aligned}$$

Finally, to compute correlation, we divide by the covariance by

$$\begin{aligned}
\sqrt{\text{Var}(\mathbf{X}_{ij}^{\text{train}})\,\text{Var}(\mathbf{X}_{ij}^{\text{test}})} &= \sqrt{\left((1-\epsilon)\Lambda_{ij} + \frac{(1-\epsilon)^2\Lambda_{ij}^2}{b_j}\right)\left(\epsilon\Lambda_{ij} + \frac{\epsilon^2\Lambda_{ij}^2}{b_j}\right)} \\
&= \sqrt{(1-\epsilon)\epsilon\Lambda_{ij}^2 + \frac{[\epsilon(1-\epsilon)]\,\Lambda_{ij}^3}{b_j} + \frac{\epsilon^2(1-\epsilon)^2\Lambda_{ij}^4}{b_j^2}} \\
&= \frac{\Lambda_{ij}}{b_j}\epsilon(1-\epsilon)\sqrt{\frac{b_j^2}{\epsilon(1-\epsilon)} + \frac{b_j\Lambda_{ij}}{\epsilon(1-\epsilon)} + \Lambda_{ij}^2}.
\end{aligned}$$

Putting it all together,

$$\begin{aligned}
\text{Cor}\left(\mathbf{X}_{ij}^{\text{train}}, \mathbf{X}_{ij}^{\text{test}}\right) &= \frac{\frac{\epsilon(1-\epsilon)\Lambda_{ij}^2}{b_j}}{\frac{\Lambda_{ij}}{b_j}\epsilon(1-\epsilon)\sqrt{\frac{b_j^2}{\epsilon(1-\epsilon)} + \frac{b_j\Lambda_{ij}}{\epsilon(1-\epsilon)} + \Lambda_{ij}^2}} \\
&= \frac{\Lambda_{ij}}{\sqrt{\frac{b_j^2}{\epsilon(1-\epsilon)} + \frac{b_j\Lambda_{ij}}{\epsilon(1-\epsilon)} + \Lambda_{ij}^2}} \\
&= \frac{\sqrt{\epsilon(1-\epsilon)}}{\sqrt{\frac{b_j^2}{\Lambda_{ij}^2} + \frac{b_j}{\Lambda_{ij}} + \epsilon(1-\epsilon)}},
\end{aligned}$$

as claimed in Proposition 3.4.2.

B.5.2 Proof of Proposition 3.4.3

Let $X_{ij} \sim \text{Poisson}\,(\gamma_i \Lambda_{ij})$. First note that

$$\begin{aligned}
\text{Cov}(\mathbf{X}_{ij}, \mathbf{X}_{ij}^{\text{train}}) &= \text{E}\left[\mathbf{X}_{ij} E\left[X_{ij}^{\text{train}} \mid X_{ij}\right]\right] - \text{E}[\mathbf{X}_{ij}] E[\mathbf{X}_{ij}^{\text{train}}] \\
&= \epsilon\, \text{E}[\mathbf{X}_{ij}^2] - \epsilon \gamma_i^2 \Lambda_{ij}^2 \\
&= \epsilon \left(\text{Var}(\mathbf{X}_{ij}) + \text{E}[\mathbf{X}_{ij}]^2\right) - \epsilon \gamma_i^2 \Lambda_{ij}^2 \\
&= \epsilon \left(\gamma_i \Lambda_{ij} + \gamma_i^2 \Lambda_{ij}^2\right) - \epsilon \gamma_i^2 \Lambda_{ij}^2 \\
&= \epsilon \gamma_i \Lambda_{ij}.
\end{aligned}$$

Next, note that $\text{SD}(\mathbf{X}_{ij}) = \sqrt{\gamma_i \Lambda_{ij}}$ and $\text{SD}(\mathbf{X}_{ij}^{\text{train}}) = \sqrt{\epsilon \gamma_i \Lambda_{ij}}$. Thus, as claimed in Proposition 3.4.3,

$$\text{Cor}(\mathbf{X}_{ij}, \mathbf{X}_{ij}^{\text{train}}) = \frac{\epsilon \gamma_i \Lambda_{ij}}{\gamma_i \Lambda_{ij} \sqrt{\epsilon}} = \sqrt{\epsilon}.$$

B.5.3 Proof of Proposition 3.4.4

Let $\mathbf{X}_{ij} \sim \text{Poisson}\left(\gamma_i \exp(\beta_{0j} + \beta_{1j} L_i)\right)$. The Fisher information matrix for the distribution of $\mathbf{X}_j$ with respect to $\beta_j = (\beta_{0j}, \beta_{1j})$ is given by

$$\mathcal{I}(\beta_j) = [1_n\ L]^T \text{diag}\left(E[\mathbf{X}_j]\right) [1_n\ L]. \tag{B.3}$$

We define $\tilde{\beta}_0 = \log(1-\epsilon) + \beta_0$ and $\tilde{\beta}_1 = \beta_1$ such that, by Proposition 3.4.1, we can write the distribution of $\mathbf{X}_{ij}^{\text{test}}$ as Poisson $\left(\gamma_i \exp(\tilde{\beta}_{0j} + \tilde{\beta}_{1j} L_i)\right)$. The Fisher Information matrix for the distribution of $\mathbf{X}_j^{\text{test}}$ with respect to $\tilde{\beta}_j = (\tilde{\beta}_0, \tilde{\beta}_1)$ is given by

$$\mathcal{I}(\tilde{\beta}_j) = [1_n\ L]^T \text{diag}\left(E[\mathbf{X}_j^{\text{test}}\right) [1_n\ L] = (1-\epsilon)[1_n\ L]^T \text{diag}\left(E[\mathbf{X}_j]\right) [1_n\ L] = (1-\epsilon)\mathcal{I}(\beta_j). \tag{B.4}$$

If we regress $\mathbf{X}_j^{\text{test}}$ on L, a Wald test for $H_0 : \tilde{\beta}_1 = 0$ is based on the approximate large sample null distribution:

$$\hat{\beta}_1\left(L, \mathbf{X}_j^{\text{test}}\right) \sim N\left(0, \left[I\left(\tilde{\beta}_j\right)^{-1}\right]_{22}\right).$$

If we regress $\mathbf{X}_j$ on L, a Wald test for $H_0 : \beta_1 = 0$ is based on the approximate large sample null distribution:

$$\hat{\beta}_1(L, \mathbf{X}_j) \sim N\left(0, \left[\mathcal{I}\left(\beta_j\right)^{-1}\right]_{22}\right).$$

Thus, using (B.4), as claimed in Proposition 3.4.4,

$$\text{Var}\left(\hat{\beta}_1\left(L, \mathbf{X}_j^{\text{test}}\right)\right) \approx \frac{1}{1-\epsilon}\,\text{Var}\left(\hat{\beta}_1(L, \mathbf{X}_j)\right).$$

B.6 Overdispersion in the cardiomyocyte data

In Section 3.6, we fit Poisson GLMs to the cardiomyocyte differentiation data from Elorbany et al. [2022]. Here, we justify the use of Poisson GLMs (and the use of count splitting) by showing that the amount of estimated overdispersion is small.

We carry out the following process using the $10,000$ cells from all 7 days of the differentiation protocol. We focus on the $p = 2,500$ high variance genes that we analyze in Section 3.6 of the main text (see Section B.7).

1. Compute $\widehat{L}(X)$, using the function $\widehat{L}(\cdot)$ used in Section 3.6 and elaborated on in Appendix B.7.

2. For $j = 1, \ldots, p$, fit a negative binomial GLM using the `MASS` package in `R` to predict X_j using $\widehat{L}(X)$. From each GLM, record the estimated overdispersion parameter $\hat{b}_j$, along with the predicted mean $\hat{\Lambda}_{ij}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, p$.

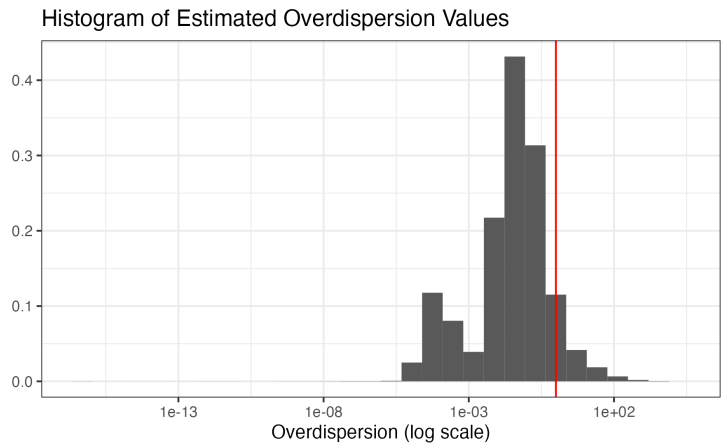


Figure B.3: Histogram of estimated values of $\frac{\Lambda_{ij}}{b_j}$ for the cardiomyocyte data. The vast majority of values are less than 1 (marked with a red vertical line).

3. Make a histogram of the $n \times p$ values of $\frac{\hat{\Lambda}_{ij}}{\hat{b}_j}$.

We note that the process above double dips in the data, and so the estimated values $\hat{\Lambda}_{ij}$ and $\hat{b}_j$ may not be completely reliable. Here, we simply use them to understand the order of magnitude of the overdispersion.

The resulting histogram is shown in Figure B.3. As noted in Section 3.4.2, the values of $\frac{\Lambda_{ij}}{b_j}$ indicate the amount of excess variance in the data. Figure B.3 shows that, for the cardiomyocyte data, the vast majority of these values are less than 1. Thus, despite the presence of a few datapoints with very large values of overdispersion, we decided to fit Poisson GLMs (which are more numerically stable than negative binomial GLMs) for the analysis in Section 3.6. As shown in Figure 3.2 of the main text, count splitting performs reasonably well when $\frac{\Lambda_{ij}}{b_j} \leq 1$.

B.7 Details of pseudotime estimation from Section 3.6

Let M be the raw gene expression matrix. We now describe the procedure used to compute $\widehat{L}(\cdot)$ in Section 3.6.

Steps 1–3 largely follow the pre-processing choices made in Elorbany et al. [2022], whereas

Steps 4–5 use the popular and user-friendly `Monocle3 R` package for trajectory estimation and are motivated by the `Monocle3` vignettes [Pliner et al., 2022].

1. **Normalization:** We create the log-normalized matrix with the (i, j)th entry given by $\log\left(\frac{M_{ij}}{\hat{\gamma}_i(M)} + 1\right)$. All operations below are run on this log-normalized matrix. The $\hat{\gamma}(\cdot)$ function is the `librarySizeFactors()` function from the `scran` package [Lun et al., 2016], which computes the row sums of M and then scales them to have a geometric mean of 1.

2. **Estimate Cell Cycle:** Using the `tricycle` package [Zheng et al., 2022], estimate the cell cycle phase for each cell using the log-normalized version of M.

3. **Feature selection:** We apply the `modelGeneVar()` function from the `scran` package. This function computes the mean and the variance of each log-normalized gene, and then fits a trend to these values. The residuals from this trend measure the biological component of variation for each gene. We then run the `getTopHVGs()` from the `scran` package, which ranks the genes by this estimated biological component and computes a p-value to test the null hypothesis that this biological component is equal to 0. We select the genes whose biological component is significantly greater than 0 with a false discovery rate cutoff of 0.01. If more than 2500 genes are deemed significant, we retain the top 2500 of these selected genes for further analysis. While this feature selection is performed separately for the training dataset, test dataset, and full dataset inside of the $\widehat{L}(\cdot)$ function, the 2500 genes whose p-values are plotted in Figure 3.4 of the main text are the top 2500 genes selected using the full dataset (rather than the training dataset or the test dataset).

4. **Preprocessing and alignment and regressing out:** We reduce the dimension of the matrix output by the previous step (log-normalized with at most 2,500 features) by keeping only the top 100 principal components of this matrix. Using the reduced

dimension matrix, we then regress out technical sources of variation using the method of Haghverdi et al. [2018], as implemented in the `Monocle3` package in the function `align_cds()`. We include cell line as the alignment group, and we additionally regress out the estimated cell cycle phase (from Step 2) and the proportion of mitochondrial reads. While this regression step is suggested in the `Monocle3` vignettes, regressing out the estimated cell cycle phase is perhaps less standard. Regressing out cell cycle is particularly important for the "day 0 only" example, as we want to ensure that no true trajectory (including a trajectory through the cell cycle) is present in the "day 0 only" data.

5. **Monocle3 dimension reduction, graph embedding, and pseudotime calculation:** Next, we compute pseudotime using a sequence of functions from the `Monocle3` package. All functions are run with their default settings unless otherwise noted. The input to this step is the matrix output by the previous step. First, we run `reduce_dimension()`, which takes the already dimension-reduced data and turns it into a two-dimensional UMAP representation. Next, we run `cluster_cells()`. (While we do not wish to estimate clusters, this is a required step of the `Monocle3` pipeline.) We next run `learn_graph()`, but we let `use_partition=FALSE` so that `Monocle3` ignores cluster information and learns a single graph through all of the cells. The clustering and graph steps both happen in UMAP space. Finally, we run `order_cells()`, which projects all cells onto the principal graph to obtain a single continuous trajectory. As `order_cells()` requires a user-specified "root" cell (the cell which will have a pseudotime of 0), we choose the induced pluripotent stem cell (IPSC) that is closest to a vertex of the graph as the "root" cell. The cell type labels (i.e. the labels that allow us to identify IPSC cells) were computed by Elorbany et al. [2022].

Appendix C

APPENDICES FOR CHAPTER 4

C.1 Implementation details for Figure 4.2

To generate Figure 4.2, we first generate a toy dataset $X \in \mathbb{Z}_{\geq 0}^{100\times 2}$, where each element X_{ij} is drawn independently from the NB$(5, 5)$ distribution, which has mean 5 and variance 10. In panels (c) and (d), the naive method that uses the data twice and our proposed method are both implemented using the details provided in Section 4.4. Here, we provide details about the implementation of sample splitting used in Figure 4.2.

We begin by splitting the cells such that the first $n/2$ cells belong to the training set, and the remaining cells belong to the test set.

We first describe Figure 4.2(c). For values of k ranging from 1 to 10, we cluster the first $n/2$ rows of the matrix $\log(X + 1)$ using k-means clustering with k clusters. This yields estimated cluster assignments $\hat{c}_i$ only for cells $i = 1, \ldots, \frac{n}{2}$ (the cells in the training set). To obtain cluster assignments $\hat{c}_i$ for the cells in the test set, we run 3 nearest neighbors classification on the log transformed training set, the log transformed test set, and the training set cluster labels. We then compute an estimated mean $\hat{\mu}_{ij}$ for each datapoint X_{ij} as

$$\hat{\mu}_{ij} = \frac{1}{\sum_{i'=1}^{n/2} \mathbf{1}\{\hat{c}'_i = \hat{c}_i\}} \sum_{i'=1}^{n/2} X_{i',j} \mathbf{1}\{\hat{c}'_i = \hat{c}_i\}, \tag{C.1}$$

which is the sample mean of the training set data points belonging to this cluster. Finally, we compute the within-cluster mean-squared error as

$$\frac{1}{n/2 \times 2} \sum_{i=n/2+1}^{n} \sum_{j=1}^{2} \left(\log(X_{ij} + 1) - \log(\hat{\mu}_{ij} + 1)\right)^2. \tag{C.2}$$

Sample splitting is implemented in a similar way in Figure 4.2(d). We run k-means clustering with $k = 2$ on the logged training data. This yields cluster assignments $\hat{c}_i$ only for $i = 1, \ldots, n/2$. We once again obtain cluster assignments $\hat{c}_i$ for $i = n/2 + 1, \ldots, n$ by applying 3 nearest neighbors to the logged training set, the logged test set, and the training set labels. Finally, for $j = 1$ and for $j = 2$, we fit a negative binomial generalized linear model where the response is X_{ij} and the covariate is $\hat{c}_i$, for $i = n/2 + 1, \ldots, n$.

In both cases, sample splitting fails because $\hat{c}_i$ for $i = n/2 + 1, \ldots, n$ is obtained using the data X_{ij} for $i = n/2 + 1, \ldots, n$, via the 3 nearest neighbors classification step. Thus, in the evaluation steps that use the test set, the test set is not truly held out data.

To create panels (e) and (f) of Figure 4.2, we apply Algorithms 4.5 and 4.6 to the single realization of toy data shown in Figure 4.2. In both algorithms, we estimate $k = 5$ clusters by running k-means on the log-transformed data. For Algorithms 4.5, we use a support vector machine (SVM) with a linear kernel as the classifier.

C.2 Proofs for Section 4.3

C.2.1 Proof of Theorem 4.2

We first state and prove the following lemma. We note that this lemma is proved much more concisely in Neufeld et al. [2023a] using the observation that the distribution of two independent negative binomial random variables conditional on their sum is beta binomial, and then applying the general data thinning proof. Here, we prove this specific result directly via algebra.

Lemma C.1 (Beta-binomial thinning of a negative binomial random variable) *Let* $\mathbf{y} \sim \mathrm{NB}(\mu, b)$ *and for any* $\epsilon \in (0, 1)$ *draw* $\mathbf{y}^{\text{train}} \mid \mathbf{y} = y \sim \mathrm{BetaBinomial}(y, \epsilon b, (1-\epsilon)b)$ *and let* $y^{\text{test}} = y - y^{\text{train}}$. *Then*

1. $\mathbf{y}^{\text{test}} \sim \mathrm{NB}(\mu, b)$

2. $\mathbf{y}^{\text{train}} \sim \mathrm{NB}(\mu, b)$

3. $\mathbf{y}^{\text{train}} \perp\!\!\!\perp \mathbf{y}^{\text{test}}$.

For ease of notation in the proof, denote $b_1 = \epsilon b$ and $b_2 = (1-\epsilon)b$. Further define $\theta = \frac{\mu}{\mu+b}$, where this parameter θ maps the mean-overdispersion parameterization of the negative binomial that is used throughout this paper back to the parameterization where $X \sim \text{NB}(\mu, b)$ is the number of successes in a series of i.i.d. Bernoulli (θ) trials until b failures occur.

Using the definition of the beta-binomial distribution,

$$Pr(\mathbf{y}^{\text{train}} = y^{\text{train}} \mid \mathbf{y} = y) = \binom{y}{y^{\text{train}}} \frac{\Gamma(y^{\text{train}} + b_1)\Gamma(y - y^{\text{train}} + b_2)\Gamma(b_1 + b_2)}{\Gamma(b_1)\Gamma(b_2)\Gamma(y + b_1 + b_2)}.$$

Now we derive the joint distribution of $\mathbf{y}^{\text{train}}$ and $\mathbf{y}^{\text{test}}$.

$$\begin{aligned}
Pr(\mathbf{y}^{\text{train}} = y^{\text{train}}, \mathbf{y}^{\text{test}} = y^{\text{test}}) &= Pr(\mathbf{y}^{\text{train}} = y^{\text{train}}, \mathbf{y} = y) \\
&= Pr(\mathbf{y}^{\text{train}} = y^{\text{train}} \mid \mathbf{y} = y) Pr(\mathbf{y} = y) \\
&= \binom{y}{y^{\text{train}}} \frac{\Gamma(y^{\text{train}} + b_1)\Gamma(y - y^{\text{train}} + b_2)\Gamma(b_1 + b_2)}{\Gamma(b_1)\Gamma(b_2)\Gamma(y + b_1 + b_2)} \frac{\Gamma(y+b)}{y!\Gamma(b)} (1-\theta)^b \theta^y \\
&= \binom{y}{y^{\text{train}}} \frac{\Gamma(y^{\text{train}} + b_1)\Gamma(y - y^{\text{train}} + b_2)\Gamma(b_1 + b_2)}{\Gamma(b_1)\Gamma(b_2)\Gamma(y + b_1 + b_2)} \frac{\Gamma(y + b_1 + b_2)}{y!\Gamma(b_1 + b_2)} (1-\theta)^{b_1 + b_2} \theta^y \\
&= \frac{1}{y^{\text{train}}!(y - y^{\text{train}})!} \frac{\Gamma(y^{\text{train}} + b_1)\Gamma(y - y^{\text{train}} + b_2)}{\Gamma(b_1)\Gamma(b_2)} (1-\theta)^{b_1} \theta^{y^{\text{train}}} (1-\theta)^{b_2} \theta^{y - y^{\text{train}}}
\end{aligned}$$

For the final step, substitute $y^{\text{test}} = y - y^{\text{train}}$.

$$\begin{aligned}
Pr(\mathbf{y}^{\text{train}} = y^{\text{train}}, \mathbf{y}^{\text{test}} = y^{\text{test}}) &= \frac{1}{y^{\text{train}}! y^{\text{test}}!} \frac{\Gamma(y^{\text{train}} + b_1)\Gamma(y^{\text{test}} + b_2)}{\Gamma(b_1)\Gamma(b_2)} (1-\theta)^{b_1} \theta^{y^{\text{train}}} (1-\theta)^{b_2} \theta^{y^{\text{test}}} \\
&= \left[\frac{\Gamma(y^{\text{train}} + b_1)}{\Gamma(b_1) y^{\text{train}}!} (1-\theta)^{b_1} \theta^{y^{\text{train}}} \right] \left[\frac{\Gamma(y^{\text{test}} + b_2)}{\Gamma(b_2) y^{\text{test}}!} (1-\theta)^{b_2} \theta^{y^{\text{train}}} \right]
\end{aligned}$$

This is the product of two independent negative binomial distributions; one for $\mathbf{y}^{\text{train}}$ and one for $\mathbf{y}^{\text{test}}$. In the success-failure parameterization, $\mathbf{y}^{\text{train}} \sim \text{NB}(b_1, \theta)$ and $\mathbf{X}^{\text{test}} \sim \text{NB}(b_2, \theta)$. This means that $\text{E}\left[\mathbf{y}^{\text{train}}\right] = \frac{\theta b_1}{1-\theta} = \epsilon \frac{\theta b}{1-\theta} = \epsilon\mu$. Similarly, $\text{E}\left[X^{(\text{test})}\right] = (1-\epsilon)\mu$. This concludes the proof of Lemma C.1.

Theorem 4.2 follows from noting that when Algorithm 4.2 is applied to a dataset X such that $\mathbf{X}_{ij} \overset{\text{ind.}}{\sim} \text{NB}(\mu_{ij}, b_j)$ with $b'_j = b_j$, then for every $i = 1, \ldots, n$, $j = 1, \ldots, p$, and $m = 1, \ldots, M$:

- $\mathbf{X}_{ij}^{\text{test,m}} \mid \mathbf{X}_{ij} = X_{ij} \sim \text{BetaBinomial}\left(X_{ij}, \epsilon_m b_j, (1 - \epsilon_m) b_j\right)$.
- $X_{ij}^{\text{train,m}} = X_{ij} - X_{ij}^{\text{test,m}}$.

Thus, the proof of Theorem 4.2 follows by applying Lemma C.1 for each $i = 1, \ldots, n$, $j = 1, \ldots, p$, and $m = 1, \ldots, M$.

C.2.2 Proof of Theorem 4.3

The first two statements in Theorem 4.3, which give the marginal distributions of $X_{ij}^{\text{train,m}}$ and $X_{ij}^{\text{test,m}}$, are relatively well-known results that can be found in, for example, Harremoës et al. [2010] and Leiner et al. [2022]. The derivation of the correlation between $\mathbf{X}_{ij}^{\text{train,m}}$ and $\mathbf{X}_{ij}^{\text{test,m}}$ is from Neufeld et al. [2022a], and is given below.

We first derive the marginal variances of $\mathbf{X}_{ij}^{\text{train,m}}$ and $\mathbf{X}_{ij}^{\text{test,m}}$ using their known con-

ditional means and variances. Recall that $\mathbf{X}_{ij}^{\text{test,m}} \mid \mathbf{X}_{ij} = X_{ij} \sim \text{Binomial}(X_{ij}, \epsilon_m)$.

$$\begin{aligned}
\text{Var}(\mathbf{X}_{ij}^{\text{test,m}}) &= \text{E}\left[\text{Var}\left[\mathbf{X}_{ij}^{\text{test,m}} \mid \mathbf{X}_{ij}\right]\right] + \text{Var}\left[\text{E}\left[\mathbf{X}_{ij}^{\text{test}} \mid \mathbf{X}_{ij}\right]\right] \\
&= \text{E}\left[(1-\epsilon_m)\epsilon_m \mathbf{X}_{ij}\right] + \text{Var}\left[\epsilon_m \mathbf{X}_{ij}\right] \\
&= (1-\epsilon_m)\epsilon_m \mu_{ij} + \epsilon_m^2 \left(\frac{\mu_{ij}^2}{b_j} + \mu_{ij}\right) \\
&= \epsilon_m \mu_{ij} + \frac{\epsilon_m^2 \mu_{ij}^2}{b_j}, \\
\text{Var}(\mathbf{X}_{ij}^{\text{train,m}}) &= \text{E}\left[\text{Var}\left[\mathbf{X}_{ij}^{\text{train,m}} \mid \mathbf{X}_{ij}\right]\right] + \text{Var}\left[\text{E}\left[\mathbf{X}_{ij}^{\text{train,m}} \mid \mathbf{X}_{ij}\right]\right] \\
&= \text{E}\left[\text{Var}\left[\mathbf{X}_{ij} - \mathbf{X}_{ij}^{\text{test,m}} \mid \mathbf{X}_{ij}\right]\right] + \text{Var}\left[(1-\epsilon_m)\mathbf{X}_{ij}\right] \\
&= \text{E}\left[\text{Var}\left[\mathbf{X}_{ij}^{\text{test,m}} \mid \mathbf{X}_{ij}\right]\right] + (1-\epsilon_m)^2 \left(\mu_{ij} + \frac{\mu_{ij}^2}{b_j}\right) \\
&= \text{E}\left[\epsilon_m(1-\epsilon_m)\mathbf{X}_{ij}\right] + (1-\epsilon_m)^2 \left(\mu_{ij} + \frac{\mu_{ij}^2}{b_j}\right) \\
&= \epsilon_m(1-\epsilon_m)\mu_{ij} + (1-\epsilon_m)^2 \left(\mu_{ij} + \frac{\mu_{ij}^2}{b_j}\right) \\
&= (1-\epsilon_m)\mu_{ij} + \frac{(1-\epsilon_m)^2 \mu_{ij}^2}{b_j}.
\end{aligned}$$

Next note that

$$\text{Var}(\mathbf{X}_{ij}^{\text{train,m}}) + \text{Var}(\mathbf{X}_{ij}^{\text{test,m}}) = \mu_{ij} + \frac{(\epsilon_m^2 + (1-\epsilon_m)^2)\mu_{ij}^2}{b_j},$$

and that since $\mathbf{X}_{ij} = \mathbf{X}_{ij}^{\text{train}} + \mathbf{X}_{ij}^{\text{test}}$,

$$\begin{aligned}
\text{Cov}\left(\mathbf{X}_{ij}^{\text{train,m}}, \mathbf{X}_{ij}^{\text{test,m}}\right) &= \frac{1}{2}\left(\text{Var}\left(\mathbf{X}_{ij}\right) - \text{Var}\left(\mathbf{X}_{ij}^{\text{train,m}}\right) - \text{Var}\left(\mathbf{X}_{ij}^{\text{test,m}}\right)\right) \\
&= \frac{1}{2}\left(\mu_{ij} + \frac{\mu_{ij}^2}{b_j} - \mu_{ij} - \frac{(\epsilon^2 + (1-\epsilon)^2)\mu_{ij}^2}{b_j}\right) \\
&= \frac{1}{2}\left(\frac{2(\epsilon_m(1-\epsilon_m))\mu_{ij}^2}{b_j}\right) = \frac{\epsilon(1-\epsilon)\mu_{ij}^2}{b_j}.
\end{aligned}$$

Finally, to compute correlation, we divide by the covariance by the square root of the product of the sample variances. Simplifying the algebra yields the result [Neufeld et al., 2022a].

C.2.3 Proof of Theorem 4.4

The first statement of Theorem 4.4 follows directly from the law of total expectation. Regardless of the value of b'_j,

$$\begin{aligned}
\mathrm{E}[\mathbf{X}_{ij}^{\mathrm{test,m}}] &= \mathrm{E}[\mathrm{E}[\mathbf{X}_{ij}^{\mathrm{test,m}} \mid \mathbf{X} = X_{ij}]] = \mathrm{E}[\epsilon_m \mathbf{X}_{ij}] = \epsilon_m \mu_{ij}, \\
\mathrm{E}[\mathbf{X}_{ij}^{\mathrm{train,m}}] &= \mathrm{E}[\mathrm{E}[\mathbf{X}_{ij}^{\mathrm{train,m}} \mid \mathbf{X} = X_{ij}]] = \mathrm{E}[(1-\epsilon_m) \mathbf{X}_{ij}] = (1-\epsilon_m) \mu_{ij}.
\end{aligned}$$

This is because the parameters b' do not affect the expected values of the DirichletMultinomial $\left(X_{ij}, \epsilon_1 b'_j, \ldots, \epsilon_m b'_j\right)$ distribution, only the variance.

The second statement of Theorem 4.4 uses the law of total variance. We start by deriving the marginal variance of $\mathrm{Var}[\mathbf{X}_{ij}^{\mathrm{test,m}}]$.

$$\mathrm{Var}(\mathbf{X}_{ij}^{\mathrm{test,m}}) = \mathrm{E}[\mathrm{Var}(\mathbf{X}_{ij}^{\mathrm{test,m}} \mid \mathbf{X}_{ij})] + \mathrm{Var}\left(\mathrm{E}\left(\mathbf{X}_{ij}^{\mathrm{test,m}} \mid \mathbf{X}_{ij}\right)\right).$$

We know that $\mathbf{X}_{ij}^{\mathrm{test,m}} \mid \mathbf{X}_{ij} \sim \mathrm{BetaBinomial}\left(\mathbf{X}_{ij}, \epsilon_m b'_j, (1-\epsilon_m) b'_j\right)$, and so we can plug in the (known) mean and variance of the beta binomial distribution.

$$\begin{aligned}
\mathrm{Var}(\mathbf{X}_{ij}^{\mathrm{test,m}}) &= \mathrm{E}\left[\frac{\mathbf{X}_{ij} \epsilon_m (1-\epsilon_m)(b'_j + \mathbf{X})}{(b'_j + 1)}\right] + \mathrm{Var}\left(\epsilon_m \mathbf{X}\right) \\
&= \frac{\epsilon_m (1-\epsilon_m) b'_j}{(b'_j + 1)} \mathrm{E}\left[\mathbf{X}\right] + \frac{\epsilon_m (1-\epsilon_m)}{(b'_j + 1)} \mathrm{E}\left[\mathbf{X}^2\right] + \epsilon_m^2 \mathrm{Var}\left(\mathbf{X}\right)
\end{aligned}$$

We also know the mean and variance of $\mathbf{X}$, since $\mathbf{X} \sim \text{NB}(\mu_{ij}, b_j)$. We plug these in and simplify.

$$\begin{aligned}
\text{Var}(\mathbf{X}_{ij}^{\text{test,m}}) &= \frac{\epsilon_m(1-\epsilon_m)b'_j\mu_{ij}}{(b'+1)} + \frac{\epsilon_m(1-\epsilon_m)}{(b'_j+1)}\left(\mu_{ij} + \frac{\mu_{ij}^2}{b_j} + \mu_{ij}^2\right) + \epsilon_m^2\left(\mu_{ij} + \frac{\mu_{ij}^2}{b_j}\right) \\
&= \epsilon_m\mu_{ij} + \frac{\epsilon_m^2\mu_{ij}^2}{b_j} + \frac{\epsilon_m(1-\epsilon_m)\mu^2}{(b'_j+1)}\left(\frac{1}{b_j}+1\right)
\end{aligned}$$

At this stage, we want to compare the magnitude of this variance to $\epsilon_m \text{Var}(\mathbf{X})$, and so we add and subtract 0.

$$\begin{aligned}
\text{Var}(\mathbf{X}_{ij}^{\text{test,m}}) &= \left[\epsilon_m\mu_{ij} + \epsilon_m\frac{\mu_{ij}^2}{b_j}\right] - \left[\epsilon_m\frac{\mu_{ij}^2}{b_j} - \epsilon_m^2\frac{\mu_{ij}^2}{b_j}\right] + \frac{\epsilon_m(1-\epsilon_m)\mu_{ij}^2}{(b'_j+1)}\left(\frac{1}{b_j}+1\right) \\
&= \epsilon_m\,\text{Var}(\mathbf{X}) - \left[\epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\right] + \frac{\epsilon_m(1-\epsilon_m)\mu_{ij}^2}{(b'_j+1)}\left(\frac{1}{b}+1\right) \\
&= \epsilon_m\,\text{Var}(\mathbf{X}) - \epsilon_m(1-\epsilon_m)\mu_{ij}^2\left(\frac{1}{b_j} - \frac{1}{(b'_j+1)}\left(\frac{1}{b_j}+1\right)\right) \\
&= \epsilon_m\,\text{Var}(\mathbf{X}) - \epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\left(1 - \frac{b+1}{b'+1}\right) \\
&= \epsilon_m\,\text{Var}(\mathbf{X}) + \epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b}\left(\frac{b_j+1}{b'_j+1} - 1\right),
\end{aligned}$$

as claimed in Theorem 4.4. We omit the derivation of $\text{Var}(\mathbf{X}_{ij}^{\text{train,m}})$, which is identical to the derivation for $\text{Var}(\mathbf{X}_{ij}^{\text{test,m}})$ and uses the fact $\text{Var}(\mathbf{X}_{ij}^{\text{train,m}} \mid \mathbf{X}_{ij} = X_{ij}) = \text{Var}(X - \mathbf{X}_{ij}^{\text{test,m}} \mid \mathbf{X}_{ij} = X_{ij}) = \text{Var}(\mathbf{X}_{ij}^{\text{test,m}} \mid \mathbf{X}_{ij} = X_{ij})$.

Finally, for the third statement of Theorem 4.4, we use the fact that we can always write

$$2 \times \text{Cov}(\mathbf{X}_{ij}^{\text{train,m}}, \mathbf{X}_{ij}^{\text{test,m}}) = \text{Var}(\mathbf{X}_{ij}) - \text{Var}(\mathbf{X}_{ij}^{\text{train,m}}) - \text{Var}(\mathbf{X}_{ij}^{\text{test,m}}).$$

We then plug in the known values of these variances. We can simplify this expression using what we derived above.

$$\begin{aligned}
2 \times \operatorname{Cov}(X^{(\text{train})}, X^{(\text{test})}) &= \operatorname{Var}(\mathbf{X}_{ij}) - \epsilon_m \operatorname{Var}(\mathbf{X}_{ij}) - \epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\left(\frac{b_j+1}{b'_j+1} - 1\right) \\
&\quad - (1-\epsilon_m)\operatorname{Var}(\mathbf{X}_{ij}) - \epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\left(\frac{b_j+1}{b'_j+1} - 1\right) \\
&= -2\epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\left(\frac{b_j+1}{b'_j+1} - 1\right) \\
\operatorname{Cov}(X^{(\text{train})}, X^{(\text{test})}) &= \epsilon_m(1-\epsilon_m)\frac{\mu_{ij}^2}{b_j}\left(1 - \frac{b_j+1}{b'_j+1}\right).
\end{aligned}$$

C.2.4 Proof of Theorem 4.5

We first derive the Fisher Information contained in X_{ij} for the parameter μ_{ij}.

$$\begin{aligned}
I_{\mu_{ij}}(\mathbf{X}_{ij}) &= -\operatorname{E}\left[\frac{d^2}{d\mu_{ij}^2}\log\left(f\left(\mathbf{X}_{ij} \mid \mu_{ij}, b_j\right)\right)\right] \\
&= -\operatorname{E}\left[\frac{d^2}{d\mu_{ij}^2}\left(\log\left(\frac{\Gamma(X_{ij}+b_j)}{\Gamma(b_j)X_{ij}!}\right) + X_{ij}\log\left(\frac{\mu_{ij}}{\mu_{ij}+b_j}\right) + b_j\log\left(\frac{b_j}{\mu_{ij}+b_j}\right)\right)\right] \\
&= -\operatorname{E}\left[X_{ij}\left(\frac{-1}{\mu_{ij}^2} + \frac{1}{(\mu_{ij}+b_j)^2}\right) + b_j\left(\frac{1}{(\mu_{ij}+b_j)^2}\right)\right] \\
&= \frac{1}{\mu_{ij}} - \frac{\mu_{ij}+b_j}{(\mu_{ij}+b_j)^2} = \frac{b_j}{\mu_{ij}(\mu_{ij}+b_j)}
\end{aligned}$$

Now we derive the Fisher Information contained in $\mathbf{X}_{ij}^{\text{test},m}$ for the parameter μ_{ij}.

$$\begin{aligned}
I_{\mu_{ij}}\left(\mathbf{X}_{ij}^{\text{test},m}\right) &= -\operatorname{E}\left[\frac{d^2}{d\mu_{ij}^2}\log\left(f\left(\mathbf{X}_{ij} \mid \mu_{ij}, b_j\right)\right)\right] \\
&= -\operatorname{E}\left[\frac{d^2}{d\mu_{ij}^2}\left(\log\left(\frac{\Gamma(X_{ij}+\epsilon_m b_j)}{\Gamma(\epsilon_m b_j)X_{ij}!}\right) + X_{ij}\log\left(\frac{\epsilon_m\mu_{ij}}{\epsilon_m\mu_{ij}+\epsilon_m b_j}\right) + \epsilon_m b_j\log\left(\frac{\epsilon_m b_j}{\epsilon_m\mu_{ij}+\epsilon_m b_j}\right)\right)\right] \\
&= -\operatorname{E}\left[X_{ij}\left(\frac{-1}{\mu_{ij}^2} + \frac{1}{(\mu_{ij}+b_j)^2}\right) + \epsilon_m b_j\left(\frac{1}{(\mu_{ij}+b_j)^2}\right)\right] \\
&= \frac{\epsilon_m}{\mu_{ij}} - \frac{\epsilon_m\mu_{ij}+\epsilon_m b_j}{(\mu_{ij}+b_j)^2} = \epsilon_m I_{\mu_{ij}}(\mathbf{X}_{ij}).
\end{aligned}$$

The statement about $I_{\mu_{ij}}\left(\mathbf{X}_{ij}^{\text{train},m}\right)$ can be derived in a similar manner, but it also follows directly from the results above and Theorem 4.2. Because Theorem 4.2 gives us independence between $\mathbf{X}_{ij}^{\text{test},m}$ and $\mathbf{X}_{ij}^{\text{train},m}$, we immediately know that $I_{\mu_{ij}}\left(\mathbf{X}_{ij}^{\text{test},m}\right) + I_{\mu_{ij}}\left(\mathbf{X}_{ij}^{\text{train},m}\right) = I_{\mu_{ij}}\left(\mathbf{X}_{ij}\right)$. The result follows.

C.3 Implementation details for Section 4.4

For a given matrix X, we use the `R` package `sctransform` in the following manner to estimate gene-specific overdispersion parameters b_j in the simulations for Section 4.2.

Briefly, `sctransform` begins by fitting a negative binomial GLM with $\mathbf{X}_j$ as the response, and the logged total number of unique molecular identifiers (UMIs) as the covariate for each gene $j = 1, \ldots, p$. This yields a maximum likelihood estimate $\hat{b}_j^{\text{MLE}}$ for each gene $j = 1, \ldots, p$. These maximum likelihood estimates are known to be quite noisy for sparse negative binomial data. Furthermore, the "null model" that only includes the total number of UMIs as a covariate should be the correct model for the majority of the genes, but will be incorrect for any genes that exhibit true differential expression across unknown latent variables. Thus, as a second step, `sctransform` fits a smooth kernel regression to estimate a relationship between the average expression of each gene and the gene-specific overdispersion. These smoothed estimates are used as the gene-specific overdispersion parameters.

We run the `vst()` function from the `sctransform` package in `R` with its default settings in Section 4.4. We note that we simulated data in which the two main assumptions of `sctransform` are met; that is, most genes are not differentially expressed, and there is a smooth relationship between the average expression of a gene and its parameter b_j. Thus, our results show a best-case scenario when it comes to using `sctransform` to estimate b_j.

C.4 Implementation details for Section 4.5

The data used in Section 4.5 comes from the publicly available dataset that is associated with Cao et al. [2020], which can be downloaded from `https://descartes.brotmanbaty.org/`. For the main cell type analysis, we used all cells from this dataset that were collected from

the kidney. For the cell subtype analysis, we used all of the kidney cells that were labeled in this dataset as Metanephric cells (i.e. were assigned Metanephric labels during the original analysis by Cao et al. [2020]). For both analyses, we filtered to genes with non-zero counts in at least 10 cells.

To carry out Step 1 of Algorithm 4.5 on our two datasets, we used the `Monocle3` package. To preprocess each dataset, we generated a 50-dimensional principal components embedding of each dataset and subsequently a 2-dimensional UMAP embedding using the default settings of the preprocessing functions in the `Monocle3` package. Next, we performed Leiden clustering using the the `Monocle3` clustering function. We chose a resolution parameter that gave a similar number of clusters to those obtained in the original paper. More specifically, we let the resolution parameter be 1×10^{-6} for the full kidney dataset and 1×10^{-5} for the metanephric cell subset. To carry out Step 2 of Algorithm 4.5, we split the 50-dimensional principal components embedding of the full count data from kidney and metanephric cells into 5 folds containing equal numbers of cells. We used the cluster labels, inferred as described above (on the full counts), as the 'true' cluster labels. For each of the 5 folds, we then trained a linear SVM model to predict the cluster assignment from 80% of the embedded expression data. We generated a confusion matrix by comparing the 'true' labels to this trained model's predictions on the held-out subset. We note that this SVM is slightly different from that of Cao et al. [2020], who trained their SVM using the whole transcriptome rather than the reduced-dimension embedding.

To carry out the "assume Poisson" version of Algorithm 4.6 on our two datasets, we began by performing Poisson count splitting (Algorithm 4.1, or, equivalently, Algorithm 4.2 with $b_j = \infty$) with $M = 2$ folds and $\epsilon_1 = \epsilon_2 = 0.5$ on each dataset to obtain an $X^{(1)}$ and an $X^{(2)}$ for each of the datasets. We then followed the preprocessing and clustering procedure outlines above on each fold for each dataset, to obtain two clusterings for each dataset. To produce the confusion matrices in Figure 4.9, we re-ordered the test set labels on the Y-axis to make the confusion matrix as diagonal as possible (since cluster 1 on the first fold does not necessarily correspond to cluster 2 on the second fold).

Finally, to carry out the negative binomial version of Algorithm 4.6 on our two datasets, we began by using the `sctranstorm` package in `R` with its default parameters to estimate overdispersion values for each gene (see Appendix C.3). We then performed negative binomial count splitting (Algoritmm 4.2) with $M = 2$ folds and $\epsilon_1 = \epsilon_2 = 0.5$ on each dataset to obtain an $X^{(1)}$ and an $X^{(2)}$ for each of the datasets. We then proceeded as in the Poisson case.

Appendix D

APPENDICES FOR CHAPTER 5

D.1 Comparison of data thinning and data fission

As mentioned in Section 5.1, the data fission proposal of Leiner et al. [2022] provides an alternate set of strategies to decompose a single realization X into $X^{(1)}$ and $X^{(2)}$. In this section, we compare and contrast the two approaches.

D.1.1 Independent decompositions

Leiner et al. [2022] provide a strategy for obtaining independent $X^{(1)}$ and $X^{(2)}$ only in the case where X is Poisson or Gaussian.

In the case of the Poisson distribution, the proposal of Leiner et al. [2022] coincides exactly with the proposal obtained from Algorithm 5.1 in this paper. This proposal is well-known, and has also been used by Oliveira et al. [2022], Sarkar and Stephens [2021], Gerard [2020], Chen et al. [2021] and Neufeld et al. [2022a].

In the case of the Gaussian distribution, the proposal of Leiner et al. [2022] has also been used by Tian and Taylor [2018], Oliveira et al. [2021], and **?**, among others. It does not follow directly from Algorithm 5.1 in this paper, since $X \neq X^{(1)} + X^{(2)}$. However, in Example D.1, we show the proposal of Leiner et al. [2022] is a simple rescaling of the proposal in this paper.

Example D.1 (Comparison of two Gaussian decompositions) *Consider the task of splitting the* $\mathrm{N}(\mu, \sigma^2)$ *distribution into two independent normally-distributed random variables, with* σ *known. The data thinning proposal is given in Table 5.2, and leads to* $X^{(1)} \sim \mathrm{N}(\epsilon\mu, \epsilon\sigma^2)$ *and* $X^{(2)} \sim \mathrm{N}((1-\epsilon)\mu, (1-\epsilon)\sigma^2)$*, where* $X^{(1)} + X^{(2)} = X$*.*

The data fission proposal is as follows: given a value of $\tau > 0$*, we draw* $Z \sim \mathrm{N}(0, \sigma^2)$*, and then let* $X^{(1)} = X + \tau Z$ *and* $X^{(2)} = X - \frac{1}{\tau}Z$*. Then,* $X' \sim \mathrm{N}(\mu, (1+\tau^2)\sigma^2)$ *and*

$X'' \sim \mathrm{N}(\mu, (1+\frac{1}{\tau^2})\sigma^2)$. *It follows that* $X' \perp\!\!\!\perp X''$. *Under this decomposition,* $X' + X'' \neq X$, *but*

$$\frac{1}{1+\tau^2}X' + \frac{\tau^2}{1+\tau^2}X'' = \frac{1}{1+\tau^2}(X+\tau Z) + \frac{\tau^2}{1+\tau^2}(X - \frac{1}{\tau}Z) = X.$$

We can easily verify that, if we let $\epsilon = \frac{1}{1+\tau^2}$, *then the random variables* $\frac{1}{1+\tau^2}X'$ *and* $\frac{\tau}{1+\tau^2}X''$ *obtained via data fission have the same distributions (both marginally and conditional on* $X = x$*) as* $X^{(1)}$ *and* $X^{(2)}$ *obtained via data thinning. For example, we can easily verify that* $\frac{1}{1+\tau^2}X' \sim N\left(\frac{1}{1+\tau^2}\mu, \frac{1}{1+\tau^2}\sigma^2\right) = N\left(\epsilon\mu, \epsilon\sigma^2\right)$. *Thus, the two decompositions are identical, up to a scaling of* $X^{(1)}$ *and* $X^{(2)}$ *by a (known) constant.*

The main idea of Example D.1 extends to the decomposition of the multivariate normal given in Table 5.2 of this paper, and the corresponding decomposition from Leiner et al. [2022].

D.1.2 Non-independent decompositions

With the exception of the Gaussian and Poisson distributions, the decompositions of Leiner et al. [2022] do not yield $X^{(1)}$ and $X^{(2)}$ that are independent. While in principle we can fit a model to $X^{(1)}$ and validate it using the conditional distribution of $X^{(2)} \mid X^{(1)}$, we will see in this section that this can be difficult to carry out in practice. In particular, we note the following drawbacks of the non-independent decompositions of Leiner et al. [2022].

(1) The distribution of $X^{(1)}$, and the conditional distribution of $X^{(2)}|X^{(1)}$, need not resemble the distribution of X. Thus, if the goal is to evaluate a potential model for X, it is not always clear what model to fit to $X^{(1)}$. We will illustrate this drawback in Example D.2.

(2) The parameters of interest are entangled in the conditional distribution of $X^{(2)} \mid X^{(1)}$. We will illustrate this issue in Example D.3.

(3) The tuning parameter that governs the information trade-off between $X^{(1)}$ and $X^{(2)}$ can be hard to interpret. For instance, in the case of the Gamma decomposition in Example D.2, the tuning parameter is $B \in \{1, 2, \ldots\}$, but in the case of the negative binomial distribution in Example D.3, it is $\epsilon \in (0, 1)$. These both contrast with Example D.1, where the tuning parameter was $\tau > 0$.

(4) The roles of $X^{(1)}$ and $X^{(2)}$ cannot be interchanged. For example, in the decomposition of the Bernoulli(θ) distribution given in Leiner et al. [2022], the distributions of $X^{(1)}$ and $X^{(2)} \mid X^{(1)}$ each contain information about θ. However, the distribution of $X^{(1)} \mid X^{(2)}$ contains no information about θ. Furthermore, while Remark 1 in Leiner et al. [2022] provides a strategy for obtaining multiple folds of training data for the data fission decompositions that are constructed using the "conjugate prior" strategy, these folds are not marginally independent of one another (they are conditionally independent given X). Beyond these specific decompositions, Leiner et al. [2022] do not provide a clear strategy for extending their decompositions to the case of multiple folds. Thus, it is not clear in general how to use data fission decompositions to carry out cross validation.

To illustrate point (1), we consider the Gamma distribution.

Example D.2 (Gamma decomposition, data fission approach) *Suppose* $X \sim \text{Gamma}(\alpha, \beta)$. *For a tuning parameter* $B \in \{1, 2, \ldots\}$, *Leiner et al. [2022] propose drawing* $Z = (Z_1, \ldots, Z_B)$, *where* $Z_i \overset{\text{ind.}}{\sim} \text{Poisson}(X)$, *and thus each* Z_i *marginally follows a* $\text{NegativeBinomial}(\alpha, 1/(\beta + 1))$ *distribution, and the* Z_i *are independent conditional on* X. *Take* $X^{(1)} = Z$, *and* $X^{(2)} = X$. *Then, the conditional distribution of* $X^{(2)} \mid X^{(1)}$ *is* $\text{Gamma}(\alpha + \sum_{i=1}^{B} Z_i, \beta + B)$.

This stands in notable contrast to Example 5.1 from the main text, in which data thinning provides independent (and Gamma-distributed) random variables.

To illustrate point (2), we revisit Application 5.2 to see a concrete example in which data thinning is straightforward but the proposal of Leiner et al. [2022] is difficult to use in practice.

Example D.3 (A comparison of negative binomial decompositions) *We observe* $X_{ij} \sim \text{NegativeBinomial}\left(r_{ij}, p_{ij}\right)$ *for* $i = 1, \ldots, n$ *and* $j = 1, \ldots, p$.

From Table 5.2, data thinning requires r_{ij} *to be known, and yields* $X_{ij}^{(1)} \sim \text{NegativeBinomial}\left(\epsilon r_{ij}, p_{ij}\right)$ *and* $X_{ij}^{(2)} \sim \text{NegativeBinomial}\left((1-\epsilon) r_{ij}, p_{ij}\right)$*, with* $X^{(1)} \perp\!\!\!\perp X^{(2)}$*,* $E[X^{(1)}] = \epsilon\, E[X]$*, and* $E[X^{(2)}] = (1-\epsilon)\, E[X]$*. Thus, as in Application 5.2 and Example 5.5,* $\hat{\mu}\left(X^{(1)}\right)$ *is an estimator for* $\epsilon\, E[X]$*, and we can evaluate* $\hat{\mu}(X^{(1)})$ *by computing the mean squared error between* $X^{(2)}$ *and* $\frac{1-\epsilon}{\epsilon}\hat{\mu}\left(X^{(1)}\right)$*.*

For $\epsilon \in (0, 1)$*, the data fission proposal of Leiner et al. [2022] draws* $X_{ij}^{(1)} \mid X_{ij} \sim \text{Binomial}(X_{ij}, \epsilon)$ *and sets* $X^{(2)} = X - X^{(1)}$*. Under this decomposition,* $X_{ij}^{(1)} \sim \text{NegativeBinomial}\left(r_{ij}, \frac{p_{ij}}{p_{ij}+\epsilon(1-p_{ij})}\right)$*, and so* $E[X^{(1)}] = \epsilon\, E[X]$*. Moreover,* $X_{ij}^{(2)} \mid X_{ij}^{(1)} \sim \text{NegativeBinomial}\left(r_{ij} + X_{ij}^{(1)}, p_{ij} + \epsilon - p_{ij}\epsilon\right)$*, and so* $E[X_{ij}^{(2)} \mid X_{ij}^{(1)}] = \left(r_{ij} + X_{ij}^{(1)}\right)\left(\frac{1-p_{ij}-\epsilon+\epsilon p_{ij}}{p_{ij}+\epsilon-\epsilon p_{ij}}\right)$*. As* $X^{(1)}$ *and* $X^{(2)}$ *are not independent, we cannot simply use mean squared error loss between* $X^{(2)}$ *and* $\frac{1-\epsilon}{\epsilon}\hat{\mu}\left(X^{(1)}\right)$ *to evaluate the estimator.*

At a glance, it might appear that an advantage of data fission over data thinning is that the former does not require knowledge of r_{ij} *to obtain* $X_{ij}^{(1)}$ *and* $X_{ij}^{(2)}$*. However, looking at the conditional distribution of* $X^{(2)} \mid X^{(1)}$*, we see that* $E[X^{(2)}]$ *is a function of* $E[X]$ *that is tractable only if* r_{ij} *is known. Thus, in practice, using the data fission approach to do inference on* $E[X]$ *will require knowing or accurately estimating the nuisance parameters* r_{ij}*.*

Similar issues arise for other decompositions given in Leiner et al. [2022].

D.2 Proofs from Section 5.2

D.2.1 Proof of Theorem 5.1

This proof is due to Joe [1996] and Jørgensen and Song [1998], but has been adapted to fit our notation.

Let x, our observed data, be a realization of a random variable $X \sim F_\lambda$. Let $\epsilon \in (0,1)$ be chosen such that $\epsilon\lambda$ and $(1-\epsilon)\lambda$ are in the parameter space Λ. We draw $X^{(1)} \mid X = x \sim G_{\epsilon\lambda,(1-\epsilon)\lambda,x}$, where this notation was defined in Section 5.2.1, and let $X^{(2)} = X - X^{(1)}$.

Separately, let $X' \sim F_{\epsilon\lambda}$ and $X'' \sim F_{(1-\epsilon)\lambda}$ be independent, and let $Y = X' + X''$. As F_λ is a convolution-closed distribution, it follows that $Y \sim F_\lambda$ and thus we know that Y has the same marginal distribution as X.

We first argue that the joint distribution of $(X^{(1)}, X)$ is the same as the joint distribution of (X', Y). The conditional distribution of $X^{(1)} \mid X$ is the same as the conditional distribution of $X' \mid Y$ by definition of the distribution $G_{\epsilon\lambda,(1-\epsilon)\lambda,x}$. Furthermore, we have already noted that the marginal distributions of X and Y are the same. Thus, the joint distribution of $(X^{(1)}, X)$ is the same as the joint distribution of (X', Y).

As $X^{(2)}$ is deterministic given $X^{(1)}$ and X, the joint distribution of $(X^{(1)}, X^{(2)})$ is the same as the joint distribution of $(X^{(1)}, X)$. Similarly, the joint distribution of (X', Y) is the same as the joint distribution of (X', X''). Thus, the joint distribution of $(X^{(1)}, X^{(2)})$ is the same as the joint distribution of (X', X''). As the joint distribution of X' and X'' is known to be the product of independent distributions $F_{\epsilon\lambda}$ and $F_{(1-\epsilon)\lambda}$, this completes the proof of parts (i) and (ii) of Theorem 5.1.

The final statement of Theorem 5.1 follows directly from Remark 5.1.

D.2.2 Proof of Proposition 5.2.1

To prove (i), note that since $X^{(1)} \mid X = x$ is normally distributed and X is normally distributed, a well-known property of the normal distribution tells us that the marginal distribution of $X^{(1)}$ is normal. We then use the law of total expectation and the law of total

variance to compute its mean and variance.

$$\begin{aligned}
\mathrm{E}[X^{(1)}] &= \mathrm{E}[\mathrm{E}[X^{(1)} \mid X]] = \mathrm{E}[\epsilon X] = \epsilon\mu \\
\mathrm{Var}(X^{(1)}) &= \mathrm{Var}\left(\mathrm{E}\left[X^{(1)} \mid X\right]\right) + \mathrm{E}\left[\mathrm{Var}\left(X^{(1)} \mid X\right)\right] \\
&= \mathrm{Var}\left(\epsilon X\right) + \mathrm{E}\left(\epsilon(1-\epsilon)\tilde{\sigma}^2\right) \\
&= \epsilon^2\sigma^2 + \epsilon(1-\epsilon)\tilde{\sigma}^2.
\end{aligned}$$

To prove (ii), note that the difference between two normally distributed variables (X and $X^{(1)}$) is normal. Then note that

$$\begin{aligned}
\mathrm{E}[X^{(2)}] &= \mathrm{E}[X] - \mathrm{E}[X^{(1)}] = \mu - \epsilon\mu = (1-\epsilon)\mu. \\
\mathrm{Var}(X^{(2)}) &= \mathrm{Var}\left(\mathrm{E}\left[X^{(2)} \mid X\right]\right) + \mathrm{E}\left[\mathrm{Var}\left(X^{(2)} \mid X\right)\right] \\
&= \mathrm{Var}\left(\mathrm{E}\left[X - X^{(1)} \mid X\right]\right) + \mathrm{E}\left[\mathrm{Var}\left(X - X^{(1)} \mid X\right)\right] \\
&= \mathrm{Var}\left((1-\epsilon)X\right) + \mathrm{E}\left[\mathrm{Var}\left(X^{(1)} \mid X\right)\right] \\
&= (1-\epsilon)^2\sigma^2 + \epsilon(1-\epsilon)\tilde{\sigma}^2,
\end{aligned}$$

which completes the proof of (ii). Finally, to prove (iii), note that

$$\begin{aligned}
2\,\mathrm{Cov}(X^{(1)}, X^{(2)}) &= \mathrm{Var}(X) - \mathrm{Var}(X^{(1)}) - \mathrm{Var}(X^{(2)}) \\
&= \sigma^2 - \epsilon^2\sigma^2 - \epsilon(1-\epsilon)\tilde{\sigma}^2 - (1-\epsilon)^2\sigma^2 - \epsilon(1-\epsilon)\tilde{\sigma}^2 \\
&= 2\epsilon(1-\epsilon)\left(\sigma^2 - \tilde{\sigma}^2\right).
\end{aligned}$$

D.2.3 Proof of Proposition 5.2.2

Recall that if $A \sim \mathrm{BetaBinomial}\,(r, \alpha, \beta)$, then $\mathrm{E}[A] = \frac{r\alpha}{\alpha+\beta}$ and $\mathrm{Var}(A) = \frac{r\alpha\beta(\alpha+\beta+r)}{(\alpha+\beta)^2(\alpha+\beta+1)}$. Then we can derive the marginal variance of $X^{(1)}$ using the law of total variance and the

fact that $X^{(1)} \mid X \sim \text{BetaBinomial}(X, \epsilon\tilde{r}, (1-\epsilon)\tilde{r})$.

$$\begin{aligned}
\text{Var}(X^{(1)}) &= \text{E}[\text{Var}(X^{(1)} \mid X)] + \text{Var}(\text{E}[X^{(1)} \mid X]) \\
&= \text{E}\left[\frac{X\epsilon(1-\epsilon)(\tilde{r}+X)}{(\tilde{r}+1)}\right] + \text{Var}\left(\epsilon X\right) \\
&= \frac{\epsilon(1-\epsilon)}{\tilde{r}+1}\left(\tilde{r}\,\text{E}\left[X\right] + \text{E}\left[X^2\right]\right) + \epsilon^2\,\text{Var}\left(X\right) \\
&= \frac{\epsilon(1-\epsilon)}{\tilde{r}+1}\left(\tilde{r}\,\text{E}\left[X\right] + \text{Var}(X) + E[X]^2\right) + \epsilon^2\,\text{Var}\left(X\right).
\end{aligned}$$

Next note that $\text{Var}(X^{(2)} \mid X) = \text{Var}(X - X^{(1)} \mid X) = \text{Var}(X^{(1)} \mid X)$ and that $\text{E}[X^{(2)} \mid X] = (1-\epsilon)X$. Thus, we arrive at:

$$\text{Var}(X^{(2)}) = \frac{\epsilon(1-\epsilon)}{\tilde{r}+1}\left(\tilde{r}\,\text{E}\left[X\right] + \text{Var}(X) + E[X]^2\right) + (1-\epsilon)^2\,\text{Var}\left(X\right).$$

To derive the covariance, note that:

$$\begin{aligned}
2\,\text{Cov}(X^{(1)}, X^{(2)}) &= \text{Var}(X) - \text{Var}(X^{(1)}) - \text{Var}(X^{(2)}) \\
&= \text{Var}(X) - 2\frac{\epsilon(1-\epsilon)}{\tilde{r}+1}\left(\tilde{r}\,\text{E}\left[X\right] + \text{Var}(X) + E[X]^2\right) - \left(\epsilon^2 + (1-\epsilon)^2\right)\text{Var}\left(X\right) \\
&= -2\frac{\epsilon(1-\epsilon)}{\tilde{r}+1}\left(\tilde{r}r\frac{1-p}{p} + r\frac{1-p}{p^2} + r^2\frac{(1-p)^2}{p^2}\right) + 2\epsilon(1-\epsilon)r\frac{1-p}{p^2} \\
&= 2\epsilon(1-\epsilon)r\frac{(1-p)^2}{p^2}\left(1 - \frac{r+1}{\tilde{r}+1}\right).
\end{aligned}$$

D.2.4 Proof of Proposition 5.2.3

The proof structure is identical to those of Proposition 5.2.1 and Proposition 5.2.2. We start by recalling that if $A \sim \text{Gamma}(\alpha, \beta)$, then $\text{E}[X] = \frac{\alpha}{\beta}$ and $\text{Var}(X) = \frac{\alpha}{\beta^2}$. Then:

$$\begin{aligned}
\text{Var}(X^{(1)}) &= E\left[\text{Var}\left(X^{(1)} \mid X\right)\right] + \text{Var}\left(\text{E}\left[X^{(1)} \mid X\right]\right) \\
&= E\left[\text{Var}\left(XZ \mid X\right)\right] + \text{Var}\left(\text{E}\left[XZ \mid X\right]\right) \\
&= E\left[X^2 \,\text{Var}\left(Z\right)\right] + \text{Var}\left(X \,\text{E}\left[Z\right]\right) \\
&= E\left[X^2 \left(\frac{\epsilon(1-\epsilon)}{\tilde{\alpha}+1}\right)\right] + \text{Var}\left(X\epsilon\right) \\
&= \frac{\epsilon(1-\epsilon)}{\tilde{\alpha}+1}\left(\text{Var}(X) + \text{E}[X]^2\right) + \epsilon^2 \,\text{Var}\left(X\right).
\end{aligned}$$

Similarly, we note that $\text{Var}(X^{(2)} \mid X) = \text{Var}(X - X^{(1)} \mid X) = \text{Var}(X^{(1)} \mid X)$, while $\text{E}(X^{(2)} \mid X) = (1-\epsilon)X$. This allows us to do a similar derivation and arrive at:

$$\begin{aligned}
\text{Var}(X^{(2)}) &= E\left[\text{Var}\left(X^{(2)} \mid X\right)\right] + \text{Var}\left(\text{E}\left[X^{(2)} \mid X\right]\right) \\
&= \frac{\epsilon(1-\epsilon)}{\tilde{\alpha}+1}\left(\text{Var}(X) + \text{E}[X]^2\right) + (1-\epsilon)^2 \,\text{Var}\left(X\right).
\end{aligned}$$

Finally,

$$\begin{aligned}
2\,\text{Cov}(X^{(1)}, X^{(2)}) &= \text{Var}(X) - \text{Var}(X^{(1)}) - \text{Var}(X^{(2)}) \\
&= \text{Var}(X) - 2\frac{\epsilon(1-\epsilon)}{\tilde{\alpha}+1}\left(\text{Var}(X) + \text{E}[X]^2\right) - \left(\epsilon^2 + 1 - 2\epsilon + \epsilon^2\right)\text{Var}(X) \\
&= -2\frac{\epsilon(1-\epsilon)}{\tilde{\alpha}+1}\left(\text{Var}(X) + \text{E}[X]^2\right) + 2\epsilon(1-\epsilon)\,\text{Var}(X) \\
&= -2\frac{\epsilon(1-\epsilon)}{\tilde{\alpha}+1}\left(\frac{\alpha(1+\alpha)}{\beta^2}\right) + 2\epsilon(1-\epsilon)\frac{\alpha}{\beta^2} \\
&= 2\epsilon(1-\epsilon)\frac{\alpha}{\beta^2}\left(1 - \frac{\alpha+1}{\tilde{\alpha}+1}\right).
\end{aligned}$$

D.3 Proof of Theorem 5.2

The proof is nearly identical to that of Theorem 5.1. It extends ideas from Jørgensen and Song [1998] and Joe [1996] to the setting of multiple folds.

Let x, our observed data, be a realization of random variable $X \sim F_\lambda$. Let $\epsilon_1, \ldots, \epsilon_M$ be chosen such that $\sum_{m=1}^{M} \epsilon_m = 1$, $\epsilon_m > 0$, and $\epsilon_m \lambda$ is in the parameter space Λ for $m = 1, \ldots, M$.

Suppose we draw $\left(X^{(1)}, \ldots, X^{(M)}\right) \mid X = x \sim G_{\epsilon_1\lambda, \epsilon_2\lambda, \ldots, \epsilon_M\lambda, x}$, where $G_{\epsilon_1\lambda, \epsilon_2\lambda, \ldots, \epsilon_M\lambda, x}$ was defined in Section 5.3.

Separately, let $X_1, X_2, \ldots, X_M$ be mutually independent random variables, where $X_m \sim F_{\epsilon_m\lambda}$, and let $Y = \sum_{m=1}^{M} X_m$. As F_λ is a convolution-closed distribution, we know that $Y \sim F_\lambda$ and thus Y has the same marginal distribution as X.

The conditional joint distribution of $(X^{(1)}, X^{(2)}, \ldots, X^{(M)}) \mid X$ is the same as the conditional joint distribution of $(X_1, X_2, \ldots, X_M) \mid Y$, by definition of the distribution $G_{\epsilon_1\lambda, \ldots, \epsilon_M\lambda, x}$. Furthermore, we have already seen that the marginal distributions of X and Y are the same. Thus, the marginal joint distribution of $(X^{(1)}, X^{(2)}, \ldots, X^{(M)})$ is the same as the marginal joint distribution of $(X_1, X_2, \ldots, X_M)$. Furthermore, by construction, the marginal joint distribution of $(X_1, X_2, \ldots, X_M)$ is that of M mutually independent random variables, where $X_m \sim F_{\epsilon_m\lambda}$. This concludes the proof of Theorem 5.2, parts 1-3. Part 4 of Theorem 5.2 follows directly from Remark 5.1.

D.4 Simulation study supporting details

In this section, we provide additional details about the simulation studies described in Section 5.4.1.

For Example 5.9, in which we select the number of principal components for binomial data, we use the following setup. For $K^* = 10$, we compute $\theta = UDV^T$ where U is a $n \times K^*$ random orthogonal matrix, D is a $K^* \times K^*$ diagonal matrix with diagonal elements equal to $5, 6, \ldots, 14$, and V is a $d \times K^*$ random orthogonal matrix. Then, $p_{ij} = \frac{\exp(\theta_{ij})}{1+\exp(\theta_{ij})}$ for

$i = 1, \ldots, n$ and $j = 1, \ldots, d$.

For Example 5.10, in which we select the number of clusters in Gamma-distributed data, we use the following setup.

In the small d, small K^* clustering setting described in Example 5.10, observations from each cluster are generated as $X_{ij} \overset{\text{ind}}{\sim} \text{Gamma}(\lambda, \theta_{c_i,j})$ where $\lambda = 20$,

$$\theta = \begin{bmatrix} 0.5 & 5 \\ 5 & 0.5 \\ 10 & 10 \\ 0.5 & 0.5 \end{bmatrix},$$

and $c_i \in \{1, 2, 3, 4\}$ is the true cluster membership for the ith observation.

In the large d, large K^* clustering setting described in Example 5.10, observations from each cluster are generated as $X_{ij} \overset{\text{ind}}{\sim} \text{Gamma}(\lambda, \theta_{c_i,j})$ where $\lambda = 2$, the $K^* \times d$ matrix θ is constructed such that for $j = 1, \ldots, d$ and $k = 1, \ldots, K^*$,

$$\theta_{kj} = \begin{cases} 0.1 & \text{if } k \leq 9 \text{ and } 10k - 9 \leq j \leq 10k + 10, \\ 1 & \text{otherwise,} \end{cases}$$

and $c_i \in \{1, 2, \ldots, 10\}$ is the true cluster membership for the ith observation.

D.5 Simulation with mean squared error loss function

D.5.1 Methods

As an alternative to the negative log-likelihood loss used in Section 5.4, here we consider applying a mean squared error loss. To do this, we simply replace the negative log likelihood from Step 4 of Algorithms 5.3 and 5.4 with the mean squared error, defined as

$$\frac{1}{nd} \sum_{i=1}^{n} \sum_{j=1}^{d} \left(X_{ij}^{(\text{test})} - \epsilon^{(\text{test})} rp_{ij}^{(K)} \right)^2 \tag{D.1}$$

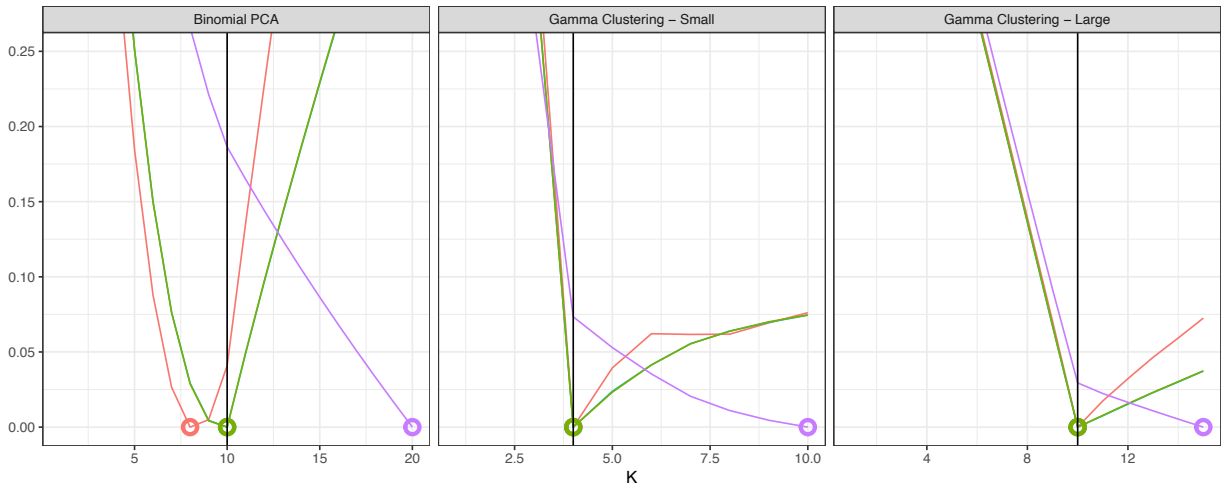


Figure D.1: The mean squared error loss averaged over 2,000 simulated data sets, as a function of K, for the naive method (purple), data thinning with $\epsilon^{(\text{train})} = 0.5$ (red), data thinning with $\epsilon^{(\text{train})} = 0.8$ (blue), and multifold thinning with 5 folds (green). Each curve has been rescaled to take on values between 0 and 1, for ease of comparison. The minimum loss values for each method are circled, and K^* is indicated by the vertical black line.

in the case of Algorithm 5.3 and

$$\frac{1}{nd}\sum_{i=1}^{n}\sum_{j=1}^{d}\left(X_{ij}^{(\text{test})} - \frac{\epsilon^{(\text{test})}}{\epsilon^{(\text{train})}}\hat{\mu}^{(\text{K})}_{\hat{c}_i,j}\right)^2 \tag{D.2}$$

in the case of Algorithm 5.4.

After replacing the loss functions, these algorithms can be applied directly to obtain simulation results for data thinning, and with slight modification to obtain results for multifold data thinning and the naive method, as described in Section 4.2.

D.5.2 Results

In Figure D.1, we plot the average mean squared error curves, as a function of K. As with the negative log-likelihood loss, data thinning approaches produce curves with sharp minimum values at or near $K = K^*$, as opposed to the naive method's monotonically-decreasing curves.

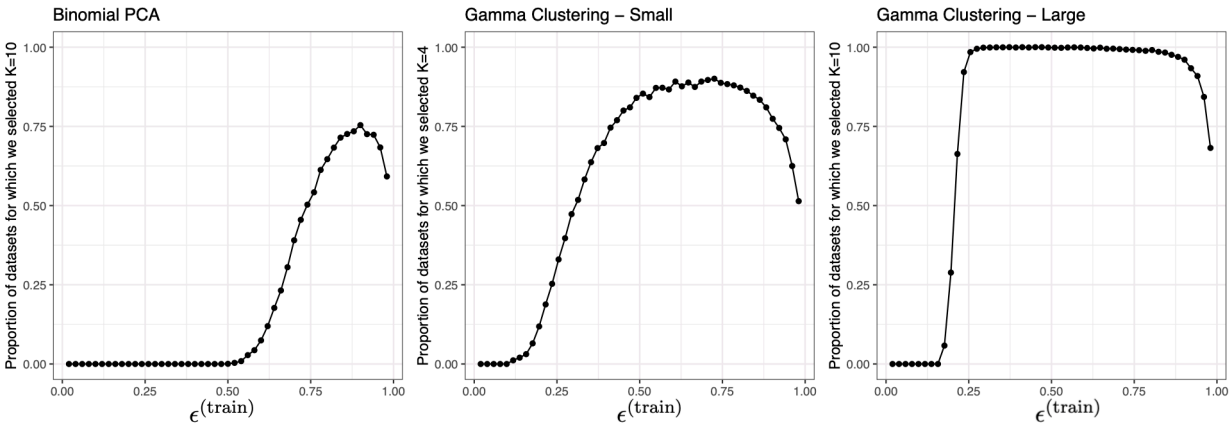


Figure D.2: The proportion of simulations for which data thinning selects the true value of K^* with the mean squared error loss, as a function of $\epsilon^{(\text{train})}$, for the simulation study described in Section 5.4.1. The optimal value of $\epsilon^{(\text{train})}$ depends on the problem at hand.

In Figure D.2, we plot the proportion of simulations that select the correct value of K^* using the mean squared error loss, as a function of $\epsilon^{(\text{train})}$. Results are largely similar to Figure 5.4.

Finally, we compare multi-fold to single-fold thinning, under the mean squared error loss, in Figure D.3. As in Figure 5.5, we find that multi-fold thinning tends to select the correct value of K more often than single-fold thinning.

D.6 Details for the real data analysis in Section 5.5

We first explain in detail the preprocessing done to the matrix X in the Seurat tutorial.

(1) Initial data filtering: We initially filter the data such that only cells with between 200 and 2500 total counts remain (with fewer than 5% of the counts coming from from mitochondrial genes) and only genes that are expressed in at least 200 cells remain. This reduces the size of X from $2,700 \times 32,738$ to $2,638 \times 13,714$

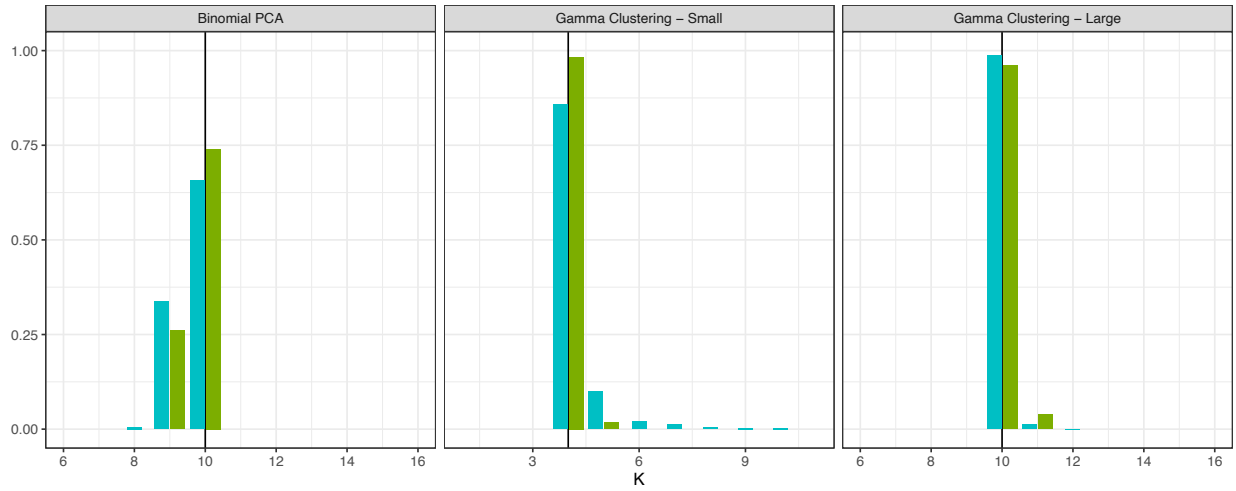


Figure D.3: The proportion of simulated data sets in which each candidate value of K is selected, with the mean squared error loss, under data thinning with $\epsilon^{(\text{train})} = 0.8$ (blue) and multifold thinning with $M = 5$ (green), for each of the simulation settings described in Section 5.4.1. The true value of K^* is indicated by the vertical black line. Multifold thinning tends to select the true value of K more often than single-fold thinning.

(2) Log normalization: Next, the data are normalized and log transformed, such that

$$Y_{ij} = \log\left(\frac{X_{ij}}{\sum_{t=1}^{13714} X_{it}} \times 10,000 + 1\right).$$

(3) Feature selection: Following this transformation, the top 2000 highly variable genes are selected using the function `FindVariableFeatures` from the Seurat package. The goal of the function is to find a subset of features with high cell-to-cell variation after accounting for the inherent mean-variance relationship, as these are most likely to be interesting in downstream analysis, and it implements methodology from Stuart et al. [2019].

(4) Centering and scaling: Finally, the columns of the subsetted matrix $Y \in \mathbb{R}^{2638 \times 2000}$ are centered and scaled to obtain the matrix $\tilde{Y}$.

After these preprocessing steps, the principal components of $\tilde{Y}$ are computed.

We now explain the preprocessing for $X^{(1)}$ and $X^{(2)}$ that we use for our data thinning alternative to the Seurat tutorial. We follow the same four steps as above, but we are careful to specify what we do on the training set $X^{(1)}$ as opposed to the test set $X^{(2)}$.

(1) Initial data filtering: We perform the initial data filtering from Step (1) above on $X^{(1)}$. We then subset $X^{(2)}$ to include the same genes and cells as those in $X^{(1)}$. After this step, $X^{(1)}$ and $X^{(2)}$ are both in $\mathbb{Z}_{\geq 0}^{2638 \times 13258}$.

(2) Log normalization: We normalize and log-transform both $X^{(1)}$ and $X^{(2)}$, such that:

$$Y_{ij}^{(1)} = \log\left(\frac{X_{ij}^{(1)}}{\sum_{t=1}^{13258} X_{it}^{(1)}} \times 10,000 + 1\right), Y_{ij}^{(2)} = \log\left(\frac{X_{ij}^{(2)}}{\sum_{t=1}^{13258} X_{it}^{(2)}} \times 10,000 + 1\right).$$

We note that these random variables are still independent and identically distributed under our Poisson assumption.

(3) Feature selection: We then apply the Seurat function `FindVariableFeatures` to the matrix $Y^{(1)}$ to select the top 2000 highly variable genes [Stuart et al., 2019]. We subset both $Y^{(1)}$ and $Y^{(2)}$ to contain only these genes, such that $Y^{(1)}, Y^{(2)} \in \mathbb{R}^{2638 \times 2000}$.

(4) Centering and scaling: We center and scale the columns of the subsetted $Y^{(1)}$ to obtain $\tilde{Y}^{(1)}$. We also center and scale the columns of the subsetted $Y^{(2)}$ to obtain $\tilde{Y}^{(2)}$.

After these preprocessing steps, the principal components of $\tilde{Y}^{(1)}$ are computed, and the loss function is computed using $\tilde{Y}^{(2)}$.

We now explain the identity that makes Figure 5.6(a) and Figure 5.6(b) mathematically equivalent. In Section 5.5, we defined

$$SSE_K(\tilde{Y}) = \left\|\tilde{Y} - U_{1:K} D_{1:K} V_{1:K}^T\right\|_F^2.$$

We see that:

$$\left\|\tilde{Y} - U_{1:K}D_{1:K}V_{1:K}^T\right\|_F^2 = \|\tilde{Y}\|_2^2 - 2\text{trace}\left(\tilde{Y}^T U_{1:K}D_{1:K}V_{1:K}^T\right) + \text{trace}\left(V_{1:K}D_{1:K}^T U_{1:K}^T U_{1:K}D_{1:K}V_{1:K}^T\right)$$
$$= \|\tilde{Y}\|_F^2 - \text{trace}\left(D_{1:K}^T D_{1:K}\right) = \left\|\tilde{Y}\right\|_F^2 - \sum_{j=1}^{K} D_{jj}^2.$$

Thus, if we compute $SSE_K(\tilde{Y})$ for $K = 1, \ldots, 20$, since $\left\|\tilde{Y}\right\|_F^2$ is fixed, we can easily obtain the values of $\sum_{j=1}^{K} D_{jj}^2$ for $K = 1, \ldots, 20$. By taking the differences between these values for K and $K+1$, we obtain D_{KK} for $K = 1, \ldots, 20$. Finally, we note that the standard deviation of the Kth principal component ($U_K D_{KK}$) can be written as

$$\sqrt{(D_{KK}U_K)^T(D_{KK}U_K)} = \sqrt{D_{KK}^2}.$$

Since the standard deviation of the Kth principal component (plotted in Figure 5.6(a)) can be obtained directly from the sums of squared error plotted in Figure 5.6(b), we say that the two plots are mathematically equivalent.