Detecting and Correcting Batch Effects in High-Throughput Genomic Experiments

Sarah Reese
Virginia Commonwealth University

Follow this and additional works at: https://scholarscompass.vcu.edu/etd
Part of the Biostatistics Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/3180
Detecting and Correcting Batch Effects in High-Throughput Genomic Experiments

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University

By

Sarah Elizabeth Reese
Bachelors of Science, University of Mary Washington, 2007

Director: Kellie J. Archer
Associate Professor, Department of Biostatistics
Director, VCU Massey Cancer Center Biostatistics Shared Resource

Virginia Commonwealth University
Richmond, Virginia
May, 2013
Acknowledgement

I wish to thank several people for their support and contributions to this work. First, I would like to express my deep gratitude to Dr. Kellie J. Archer, my dissertation advisor, and Dr. Jeanette E. Eckel-Passow for their guidance, encouragement, and useful critiques of my research. I would also like to thank the rest of my dissertation committee, Dr. Chris Gennings, Dr. Nitai Mukhopadhyay, and Dr. Sarah Elsea for their support and comments. I would especially like to thank Dr. Celine Vachon and Dr. Mariza de Andrade for allowing me to use their data in my research.

I would additionally like to thank the administrative staff of the Department of Biostatistics at VCU, especially Yvonne Hargrove, whose support was invaluable throughout my time at VCU.

Finally, I wish to thank my fiancé, Elliott, my parents, grandparents, and all my friends for their love, support and patience over the years.
Table of Contents

Table of Contents iii
List of Figures vi
List of Tables viii
Abstract x

1 Introduction 1
  1.1 Batch Effects .......................... 1
  1.2 Methods of Batch Effect Identification .................... 2
      1.2.1 Hierarchical Clustering ......................... 2
      1.2.2 Principal Components Analysis (PCA) ............... 4
      1.2.3 Correlation .................................. 5
  1.3 Methods of Batch Effect Correction ..................... 7
      1.3.1 Global Normalization .......................... 7
      1.3.2 Frozen Robust Multiarray Analysis (fRMA) ............ 9
      1.3.3 Orthogonal Projections to Latent Structures (OPLS) .... 10
      1.3.4 Corrected Robust Linear Models with Maximum Likelihood Classification Version 2 (CRLMMv2) ................... 11
      1.3.5 Batch Mean Centering (BMC) ..................... 14
      1.3.6 Distance Weighted Discrimination (DWD) ............. 15
      1.3.7 Empirical Bayes (EB) .......................... 18
      1.3.8 Surrogate Variable Analysis (SVA) ................. 22
  1.4 Evaluation Method ................................ 27
  1.5 Guided Principal Component Analysis (gPCA) ............. 28

2 Statistical Methods 29
  2.1 Principal Components Analysis (PCA) ................... 29
  2.2 Guided PCA ....................................... 31
  2.3 Proposed Method: Test statistic for testing if batch effect exists ........ 32

3 Simulation Study 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Description of Simulation Study</td>
<td>34</td>
</tr>
<tr>
<td>3.1.1 Evaluating Type I Error</td>
<td>34</td>
</tr>
<tr>
<td>3.1.2 Evaluating Power</td>
<td>34</td>
</tr>
<tr>
<td>3.1.3 Simulating Phenotypic Effects</td>
<td>35</td>
</tr>
<tr>
<td>3.2 Results</td>
<td>37</td>
</tr>
<tr>
<td>3.2.1 Dependent Batch and Phenotype Effects</td>
<td>39</td>
</tr>
<tr>
<td>3.2.2 Varied Batch Variance and Phenotypic Means Greater than Batch Means</td>
<td>39</td>
</tr>
<tr>
<td>3.2.3 High Proportion of Features Affected by Batch</td>
<td>43</td>
</tr>
<tr>
<td>3.2.4 Sensitivity of gPCA Results to Filtering of Simulation Data</td>
<td>43</td>
</tr>
<tr>
<td>3.2.5 Analysis of Varying Batch Sample Size</td>
<td>46</td>
</tr>
<tr>
<td>4 Applications</td>
<td>50</td>
</tr>
<tr>
<td>4.1 Data</td>
<td>50</td>
</tr>
<tr>
<td>4.1.1 Filtering</td>
<td>50</td>
</tr>
<tr>
<td>4.1.2 GENEMAM</td>
<td>51</td>
</tr>
<tr>
<td>4.1.3 GENOA</td>
<td>51</td>
</tr>
<tr>
<td>4.2 Results</td>
<td>52</td>
</tr>
<tr>
<td>4.2.1 GENEMAM</td>
<td>52</td>
</tr>
<tr>
<td>4.2.2 GENOA</td>
<td>55</td>
</tr>
<tr>
<td>4.2.3 Sensitivity of gPCA Results to Filtering</td>
<td>57</td>
</tr>
<tr>
<td>4.2.4 gPCA Run Time Analysis</td>
<td>61</td>
</tr>
<tr>
<td>4.2.5 Analysis of Batch and Phenotype Confounding</td>
<td>61</td>
</tr>
<tr>
<td>5 Comparison of Batch Effect Adjustment Methods</td>
<td>63</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>63</td>
</tr>
<tr>
<td>5.2 Statistical Methods</td>
<td>64</td>
</tr>
<tr>
<td>5.2.1 Guided Principal Components Analysis</td>
<td>64</td>
</tr>
<tr>
<td>5.2.2 Batch Mean-Centering</td>
<td>65</td>
</tr>
<tr>
<td>5.2.3 Distance Weighted Discrimination</td>
<td>66</td>
</tr>
<tr>
<td>5.2.4 Empirical Bayes</td>
<td>70</td>
</tr>
<tr>
<td>5.2.5 Evaluation of Batch Effect Correction Methods</td>
<td>73</td>
</tr>
<tr>
<td>5.3 Application Data</td>
<td>74</td>
</tr>
<tr>
<td>5.3.1 Simulation Study</td>
<td>74</td>
</tr>
<tr>
<td>5.3.2 Case Studies</td>
<td>75</td>
</tr>
<tr>
<td>5.4 Results</td>
<td>77</td>
</tr>
<tr>
<td>5.4.1 Batch Correction Sensitivity Analysis</td>
<td>78</td>
</tr>
<tr>
<td>5.5 Discussion</td>
<td>80</td>
</tr>
<tr>
<td>6 The gPCA Package for Identifying Batch Effects</td>
<td>81</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Average Linkage Hierarchical clustering plot using $1 - \rho$ as the dissimilarity measure between samples of the GENEMAM data. Branches are labeled with plate numbers for each sample. ................................................................. 3

1.2 GENEMAM - Unguided PCA of $X$. Samples for each plate are denoted by a different color and/or symbol. ................................................................. 6

3.1 Power for detecting batch effect as a function of the proportion of features that are affected by batch when no true phenotype was included with batch proportion ranging from 0.1 to 1%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$. ................................................................. 40

3.2 Power for detecting batch effect as a function of the proportion of features that are affected by batch when a true high variance phenotype was included in the data with batch proportion ranging from 0.1 to 2.5%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$. ................................................................. 41

3.3 Power for detecting batch effect as a function of the proportion of features that are affected by batch when a true low variance phenotype was included in the data with batch proportion ranging from 0.1 to 2.5%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$ and the proportion of features affected by phenotype is either $pprop = 0.1$ or $pprop = 0.05$. ................................................................. 42

3.4 Cumulative variance of the principal components from unguided and guided PCA. ................................................................. 43

3.5 Power for detecting batch effect as a function of the proportion of features that are affected by batch when a true phenotype was included in the data and batch and phenotype effect are not independent. The batch proportion ranges from 0.1 to 1%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$ and the proportion of features affected by phenotype is either $pprop = 0.1$ or $pprop = 0.05$. The parameters $\beta_p = 0.5$ and $\beta_b = 1$ ................................................................. 45

3.6 Power plots while varying the variance associated with batch and the phenotype means. ................................................................. 49

4.1 GENEMAM - (a) Unguided PCA of $X$ of $Y'X$. Samples for each plate are denoted by a different color and/or symbol. ................................................................. 53

4.1 GENEMAM - (b) Guided PCA of $Y'X$. Samples for each plate are denoted by a different color and/or symbol. ................................................................. 54
4.2 GENEMAM - Standardized Heatmaps showing the (a) PC$_1$ values at each sample well location. White spaces indicate missing samples for the plate. Plates 5 and 8 were incomplete plates. 55
4.2 GENEMAM - Standardized Heatmaps showing the (b) PC$_2$ values at each sample well location. White spaces indicate missing samples for the plate. Plates 5 and 8 were incomplete plates. 56
4.3 GENOA - (a) Unguided PCA of $X$. Samples for each plate are denoted by a different color and/or symbol. 59
4.3 GENOA - (b) Guided PCA of $Y'X$. Samples for each plate are denoted by a different color and/or symbol. 60
6.1 Distribution plot of $\delta_p$ values . 86
6.2 Principal components plot of first two principal components from gPCA . 87
6.3 Principal components plots of the first three principal components with density plots of the principal components on the diagonal. 88
List of Tables

1.1 Methods of batch effect correction, implementation and platform of those methods. ........................................... 8

3.1 Type I Error Simulation Scenarios: \( n = 90 \) and \( p = 1000 \) for all scenarios. \( \mu_{b_1} \) and \( \mu_{b_2} \) are the batch means, \( \sigma_b \) is the variance associated with batch, \( \mu_{p_1} \) and \( \mu_{p_2} \) are the phenotypic means, \( \sigma_p \) is the variance associated with phenotype, and \( \text{pprop} \) is the proportion of features effected by phenotype. ............. 35

3.2 Power Simulation Scenarios: \( n = 90 \) and \( p = 1000 \) for all scenarios. \( b \) denotes the number of batches, \( n_1 \) and \( n_2 \) give the number of samples in each batch, \( \mu_{b_1} \) and \( \mu_{b_2} \) are the batch means, \( \sigma_b \) is the variance associated with batch, \( \mu_{p_1} \) and \( \mu_{p_2} \) are the phenotypic means, \( \sigma_p \) is the variance associated with phenotype, and \( \text{pprop} \) is the proportion of features effected by phenotype. ............. 36

3.3 Estimated Type I Error: For all scenarios there is no true batch effect. Scenario (a) has no phenotypic effect in the data, however scenario (b) has a high variance phenotypic effect included in the analysis with phenotypic effect at \( \text{pprop} = 0.1 \) and scenarios (c-d) have low variance phenotypic effects included in the analysis with phenotypic effect at \( \text{pprop} = 0.1 \) or 0.05, respectively. . . 39

3.4 Power for detecting batch effect as a function of the proportion of features that are affected by batch at 50 to 90% when no phenotypic, high variance phenotypic, or low variance phenotypic data were included in gPCA. . . 44

3.5 Variance Filtering Sensitivity Results: \( \delta \), corresponding \( p \)-values resulting from retaining between 10 and all features from the simulation data sets. The last column gives the system time in minutes required to run gPCA as discussed in Section 4.2.4. ................................. 47

3.6 Test statistic \( \delta \) and corresponding \( p \)-value resulting from varying the batch sample sizes between 12 and 84. \( \delta_s \) and \( p_s \) indicate the results of the test when considering the batch effect indicator matrix \( Y \) as scaled by batch sample size \( n_k \). .................. 48

4.1 Variance Filtering Sensitivity Results: \( \delta \), corresponding \( p \)-values resulting from retaining between 10 and all features from the full data set. The last column gives the system time in minutes required to run gPCA as discussed in Section 4.2.4. ................................. 58

4.2 Number of features retained using an ANOVA filtering method with different multiple comparison adjustment methods and stringencies. .................. 61
4.3 Results of Confounding Batch and Phenotype: Number of significant features prior to and post-batch correction using BMC when batch and phenotype are confounded. Rows give the results of the test prior to batch correction and the columns give results of the test post-batch correction. The “Reject” column and row indicate the number of features that are significantly predicted by phenotype.

5.1 Comparison of Batch Correction Methods: test statistic $\delta$ and corresponding $p$-values before and after batch correction for the three simulated data scenarios with no phenotypic effect, high variance phenotypic effect, and low variance phenotypic effect, and the two case study data sets, GENEMAM and GENOA. Batch correction methods used are (b) batch mean centering, (c) multiclass distance weighted discrimination, (d) non-parametric empirical Bayes, and (e) parametric empirical Bayes. Test results for the uncorrected data are given in column (a). A ‘NA’ indicates that that batch correction method was not possible for that data due to no phenotypic variable available.

5.2 Run Time Analysis of Batch Correction Methods. Time is in seconds unless otherwise noted. Batch correction methods used are (a) batch mean centering, (b) multiclass distance weighted discrimination, (c) non-parametric empirical Bayes, and (d) parametric empirical Bayes. A ‘NA’ indicates that that batch correction method was not possible for that data due to no phenotypic variable available.

5.3 Contingency tables from simulated data with dependent batch and phenotypic effects that show the number of features truly significant versus those found to be significant using lmFit() and eBayes() on (a) raw data, (b) batch corrected data using batch mean-centering (BMC), (c) batch corrected data using non-parametric empirical Bayes (EBn), and (d) batch corrected data using parametric empirical Bayes (EBp). The rows of the tables indicate truth and the columns indicate the test results.
Abstract

DETECTING AND CORRECTING BATCH EFFECTS IN HIGH-THROUGHPUT GENOMIC EXPERIMENTS

By Sarah Elizabeth Reese

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University

Virginia Commonwealth University, 2013

Director: Kellie J. Archer, Ph.D., Associate Professor, Department of Biostatistics; Director, VCU Massey Cancer Center Biostatistics Shared Resource

Batch effects are due to probe-specific systematic variation between groups of samples (batches) resulting from experimental features that are not of biological interest. Principal components analysis (PCA) is commonly used as a visual tool to determine whether batch effects exist after applying a global normalization method. However, PCA yields linear combinations of the variables that contribute maximum variance and thus will not necessarily detect batch effects if they are not the largest source of variability in the data. We present an extension of principal components analysis to quantify the existence of batch effects, called guided PCA (gPCA). We describe a test statistic that uses gPCA to test if a batch effect exists. We apply our proposed test statistic derived using gPCA to simulated data and to two copy number variation case studies: the first study consisted of 614 samples from a breast cancer family study using Illumina Human 660 bead-chip arrays whereas the
second case study consisted of 703 samples from a family blood pressure study that used
Affymetrix SNP Array 6.0. We demonstrate that our statistic has good statistical properties
and is able to identify significant batch effects in two copy number variation case studies.
We further compare existing batch effect correction methods and apply gPCA to test their
effectiveness. We conclude that our novel statistic that utilizes guided principal components
analysis to identify whether batch effects exist in high-throughput genomic data is effective.
Although our examples pertain to copy number data, gPCA is general and can be used on
other data types as well.
1 Introduction

1.1 Batch Effects

Batch effects are defined to be systematic non-biological variation between groups of samples (or batches) due to experimental artifacts [2, 8, 22, 31, 35]. Many factors contribute to the generation of batch effects. Some of these include chip type, platform, lab, technician, storage and shipment conditions, protocols (which include sample extraction, amplification, labeling, and hybridization methods), cRNA/cDNA synthesis, wash conditions, etc. [31]. Often, ‘batch’ is a term that represents that a group of microarrays were processed at the same time, by the same technician, in the same lab, or with the same materials [8, 22, 25, 26]. Due to the scale of microarray experiments and the limitations of microarray technology, batch effects are unavoidable [8], but out of the thousands of microarray papers that are published every year, few actually address the problem of batch effects [8] and even fewer use a method to detect whether their data includes effects due to batch.

An early literature reference pertaining to batch effects in array studies was an abnormality affecting uniformity and reproducibility of fluorescent signal discovered in DNA microarrays [14]. The researchers observed that the quality of the batch-processed arrays was correlated to environmental ozone levels during posthybridization array washing. They measured fluorescent intensity and ratio reproducibility to determine the effect of ozone on the arrays.
1.2 Methods of Batch Effect Identification

There are few methods that have been developed to statistically test for batch effects. For expression data, methods include hierarchical clustering [2, 9, 22, 24], Pearson’s correlation [5], and principal components analysis/singular value decomposition (PCA/SVD) [16, 41]. Johnson et al. [22], Konstantinopoulos et al. [24], and Chow et al. [9] used unsupervised hierarchical clustering to identify the batch effects, prior to using an empirical Bayes framework (of Johnson et al. [22]) to adjust for batch effects. Bylesjö et al. [5] use Pearson correlation to identify potential batch effects. Alter et al. [1], Yang et al. [41], and Holmes et al. [16] use methods based on singular value decomposition or principal components analysis to detect batch effects. Each method is described in the following subsections. These methods are limited in that they do no provide a statistical test for batch effects.

1.2.1 Hierarchical Clustering

Hierarchical clustering is one of the most common methods used to detect batch effects. The algorithm is applied to the microarray data and if the data appears to group according to a potential batch effect source, like date or lab, then it is concluded that batch effect should be accounted for in downstream analysis [26]. Eisen et al. [13] and Lazar et al. [26] discuss how hierarchical clustering methods are useful in gene expression data analysis. There are two general classes of clustering methods, supervised clustering in which samples or features are clustered with respect to known phenotypic features, and unsupervised clustering in which we have no a priori phenotypic knowledge or choose not to account for any known phenotype in the analysis [13]. In most cases unsupervised hierarchical clustering is used for batch effect detection because we do not know if there are batch effects or what could be causing them.
Commonly, the clustering results are visualized by plotting a dendrogram which shows homogenous groups in which samples cluster [13, 26]. Dendrograms represent the relationship between samples with a tree-like structure where the branch lengths indicate the degree of similarity between the samples as assessed by the clustering algorithm. The GENEMAM data consists of 614 samples across 8 96-well Illumina 660 plates. In Figure 1.1, a dendrogram is displayed where average linkage hierarchical clustering was applied to the GENEMAM data set using one minus the correlation, $1 - \rho$, as the dissimilarity measure between features. Branches are labeled using the plate number for each sample. Plates 1-4, plate 5, and plates 6-8 form clusters in this plot which corresponds to the run time of these plates.

Figure 1.1: Average Linkage Hierarchical clustering plot using $1 - \rho$ as the dissimilarity measure between samples of the GENEMAM data. Branches are labeled with plate numbers for each sample.

Johnson et al. [22] used the “standard [average linkage] hierarchical clustering algorithm
produced using the dChip software [30]” to show that the samples in their data grouped by batch “indicating that the clustering algorithm recognized the batch-to-batch variation as the most significant source of variation” in the data set [22]. Johnson et al. [22] produced heatmaps with dendrograms for their raw data, after standardizing within batch, and after applying empirical Bayes batch adjustments. Konstantinopoulos et al. [24] combined multiple data sets and used hierarchical clustering on this combined training data set to show that their combined data separated by data set prior to batch adjustment and after batch adjustment samples from the different data sets were mixed. Chow et al. [9] used average linkage hierarchical clustering to assess their data for batch effects and found notable batch effects. In all three of the previous experiments, the authors used the empirical Bayes framework of Johnson et al. [22] to adjust their data for batch effects (see 1.3.7 below). Benito et al. [2] used hierarchical clustering both before batch adjustment to identify batches and after batch adjustment with their method distance weighted discrimination (DWD; see 1.3.6 below) to verify that their adjustment method removed the batch effects.

1.2.2 Principal Components Analysis (PCA)

A common method for visualizing the existence of batch effects is principal components analysis (PCA), and as the numerical workhorse of PCA, singular value decomposition (SVD). PCA is a form of unsupervised learning used for data reduction and interpretation. It looks for the linear combination of variables (probes, genes, features, etc. in genomic data) that explain the greatest variation in the data. The first two principal components are plotted with each sample colored by the suspected batch and separation of colors is taken as evidence of a batch effect. Figure 1.2 shows an example PCA plot using the GENEMAM data. In
this case study, batch is the largest source of variation in the data, therefore the PCA plot separates the plates based on time of analysis.

Yang et al. [41] determined by looking at PCA plots that the batch effects in their data were confounded with the experimental factor, and they could not be removed. To adjust for batch effects, Holmes et al. [16] redid part of their experiment using only one protocol in data collection for samples that were identified using PCA as having batch effects. In their analysis, they paired experimental and control samples to minimize experimental bias.

However, as pointed out by Benito et al. [2], if the batch effect is not the greatest source of variation then PCA methods do not work well since they look for the directions of greatest variation. The SVD/PCA approach can easily fail when variation due to systematic bias is similar or smaller than variation due to other experimental effects. Also, visual inspection of the first and second principal components is subjective. Thus, methods that can detect batch effects are needed as ignoring the potential for batch effects can have a serious effect on downstream analysis results. Although various methods have been used to detect the presence of batch effects, they are largely subjective. In this thesis, an inferential testing framework for detecting the presence of batch effects is proposed and evaluated.

1.2.3 Correlation
To quantify the batch effects in a two-channel microarray experiment to specifically account for the array bias in their data, Bylesjö et al. [5] used the Pearson correlation coefficient between the $Y$-orthogonal score vector and the average $A$ values, where $Y$ is the response matrix containing, for example, phenotypic data, and $A = \log_2(\sqrt{RG})$ where $R$ and $G$ are the red and green fluorescence intensities, respectively. To identify potential array-dye or
Figure 1.2: GENEMAM - Unguided PCA of $X$. Samples for each plate are denoted by a different color and/or symbol.
array-spatial interaction effects, they looked at the corresponding loading vector for systematic trends.

1.3 Methods of Batch Effect Correction

Although a few subjective methods for detecting the presence of batch effects have been described, several methods have been developed to correct for batch effects. These include various normalization techniques [1, 5, 39], frozen robust multiarray analysis (fRMA) [29, 33], orthogonal projections to latent structures (OPLS) [5], corrected robust linear models with maximum likelihood classification (CRLMMv2) [6], prediction analysis for microarrays or batch mean centering (PAMR/BMC) [31, 38, 40], distance weighted discrimination (DWD) [2, 17, 18, 32], empirical Bayes [9, 22, 24, 29, 39], and surrogate variable analysis (SVA) [27, 29]. Additionally, Luo et al. [31] looked at the impact of batch effect removal on cross-batch prediction performance and Lazar et al. [26] and Chen et al. [8] provided surveys of some of the many methods of batch effect removal. Below I provide a brief description of the aforementioned methods to correct for batch effects. Table 1.1 provides sources that implement the various correction methods and the platforms they used.

1.3.1 Global Normalization

Sun et al. [39] evaluated three common global normalization methods and investigated their performance with respect to batch effect removal using three human methylation datasets (Illumina HumanMethylation27 BeadChips) with different degrees of batch effects. The three global normalization methods they assessed were quantile normalization at average $\beta$ value (QN$\beta$), two step quantile normalization at probe signals (lumi), and quantile normalization of A and B signal separately (ABnorm). Sun et al. [39] found that the three methods
Table 1.1: Methods of batch effect correction, implementation and platform of those methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Implementation</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empirical Bayes (EB or ComBat) [22]</td>
<td>[9, 24, 29, 39]</td>
<td>Microarray, CNV</td>
</tr>
<tr>
<td>Distance Weighted Discrimination (DWD) [32]</td>
<td>[2, 17, 18]</td>
<td>Microarray</td>
</tr>
<tr>
<td>Batch-mean Centering (BMC) [38]</td>
<td>[31, 38, 40]</td>
<td>Microarray</td>
</tr>
<tr>
<td>Frozen Robust Multiarray Analysis (fRMA) [33]</td>
<td>[29, 33]</td>
<td>Microarray (Affymetrix</td>
</tr>
<tr>
<td>Corrected Robust Linear Models with Maximum</td>
<td>[6]</td>
<td>SNP, microarray</td>
</tr>
<tr>
<td>Likelihood Classification (CRLMM) [6]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surrogate Variable Analysis (SVA) [27]</td>
<td>[28, 29]</td>
<td>Expression Microarray</td>
</tr>
</tbody>
</table>

could remove a portion of the batch effects and their effectiveness differed depending on the severity of the batch effects; however, all methods left substantial batch effects intact in the datasets with obvious batch effects and further correction was necessary. Empirical Bayes (EB) batch adjustment [22] (see 1.3.7 and chapter 5 for more details) was subsequently evaluated which successfully removed the remaining non-biological effects and thus Sun et al. [39] recommend EB correction along with global normalization procedures for effective batch effect removal. Standard global normalization procedures are not enough to remove variation due to batch effects since they only account for global effects, that is, they correct for sources of variability that affect all probes similarly [22, 26]. Batch effects that exist after global normalization has been performed are likely due to probe-specific effects and therefore require an additional normalization step.

To correct for batch effects, Alter et al. [1] used normalization to filter out the batch effect indicative eigengenes that were found using SVD. Alter et al. [1] define SVD as “a
linear transformation of the expression data from the genes \( \times \) arrays space to the reduced ‘eigengenes’ \( \times \) ‘eigenarrays’ space.” They used the Shannon entropy, \( e \), to determine whether one eigengene captures all the expression in the dataset (\( e = 0 \)) or all eigengenes are equally expressed (\( e = 1 \)), where \( e \) is calculated as

\[
0 \leq e = -\frac{1}{\log(Q)} \sum_{q=1}^{Q} p_q \log(p_q) \leq 1
\]

and \( Q = \min(n, p) \) and \( p_q = \frac{\lambda_q^2}{\sum_{q=1}^{Q} \lambda_q^2} \) is the average of the squared singular values, \( \lambda \), resulting from the SVD on the data \( \mathbf{X} \) and \( n \) and \( p \) are the number of samples and features, respectively. Alter et al. [1] then normalized the data by filtering out the eigengenes that represent noise or experimental artifacts according to the entropy measure by substituting zero for singular values found to represent noise, \( \lambda_q = 0 \), and reconstructing the data \( \mathbf{X} \) according the the SVD equation \( \mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}' \) where \( \lambda_q \) are the non-zero diagonal entries of \( \mathbf{D} \).

### 1.3.2 Frozen Robust Multiarray Analysis (fRMA)

McCall et al. [33] present a new method based on Robust Multiarray Analysis (RMA) [20] which performs background correction, global normalization, and summarization in a modular way for gene expression microarrays. RMA has many benefits, however it is dependent on multiple arrays being analyzed simultaneously preventing it from being used in clinical settings where samples are processed individually or in small batches and data sets that are preprocessed separately are not comparable. A similar preprocessing algorithm called frozen RMA (fRMA) is presented which allows microarrays to be analyzed individually or in small batches and combined for analysis. fRMA provides similar background correction,
normalization, and summarization steps as RMA, however it accounts for between-probe and between-batch variability in the summarization step that allows single arrays or small batches of arrays to be processed. This method was developed specifically for use with Affymetrix GeneChips. McCall et al. [33] found “that fRMA is comparable to RMA when the data are analyzed as a single batch and outperforms RMA when analyzing multiple batches.” Leek et al. [29] adapted fRMA in their R package sva to remove latent variation in data and in samples obtained in future studies which Leek et al. [29] called frozen SVA (fSVA).

1.3.3 Orthogonal Projections to Latent Structures (OPLS)

Bylesjö et al. [5] introduced a multivariate latent variable regression method to identify and discard various forms of systematic bias using orthogonal projections to latent structures (OPLS). Bylesjö et al. [5] present a normalization strategy for multi-channel microarray data which utilizes a multivariate regression method to identify and discard various forms of systematic bias using OPLS. OPLS “identifies joint variation within biological samples to enable removal of sources of variation that are mathematically independent (orthogonal) to the within-sample variation. This ensures that systematic variation related to the underlying biological samples is separated from the remaining, bias-related sources of structured variation.” OPLS uses information from a response matrix $Y$ to decompose a data matrix $X$ into correlated, orthogonal, and residual structures of information such that

$$X = T_p P_p^T + T_o P_o^T + E$$

where $T_p$ is the predictive score matrix for $X$, $P_p^T$ is the predictive loading matrix for $X$, $T_o$ is the $Y$-orthogonal score matrix, $P_o^T$ is the loading matrix of the $Y$-orthogonal components,
and $\mathbf{E}$ is the residual matrix of $\mathbf{X}$. The data $\mathbf{X}$ are then normalized by removing the structured variation that is portrayed by $\mathbf{T}_p\mathbf{P}_o^T$, leaving the normalized data $\mathbf{X}^* = \mathbf{T}_p\mathbf{P}_p^T + \mathbf{E}$ which represents the biological variation in the data. Bylesjö et al. [5] apply their method to publicly available dual-channel microarray data. OPLS performs well when compared to various other global normalization methods including global median, global loess, print-tip loess, and global loess with ANOVA.

1.3.4 Corrected Robust Linear Models with Maximum Likelihood Classification Version 2 (CRLMMv2)

In an analysis of Genome Wide Association Study (GWAS) data in which single nucleotide polymorphism (SNPs) were examined, Carvalho et al. [6] found that “variability in microarray output quality across different SNPs, different arrays, and different sample batches have substantial influence on the accuracy of genotype calls made by existing algorithms. Failure to account for these sources of variability can adversely affect the quality of findings reported by the GWAS.” For SNP/copy number data, the authors developed the corrected robust linear models with maximum likelihood classification (CRLMM) version 2 to enhance their existing method, the multi-level model used by CRLMM version 1 (CRLMMv1), to account for variability across batches which allows for the identification of low-quality SNPs, samples and batches.

CRLMMv1 defines a training set using HapMap calls from known genotypes and then uses a two-stage hierarchical model for a supervised learning approach. CRLMMv1 defines $M \equiv \log_2(I_A/I_B)$ where $I_A$ and $I_B$ are the summarized intensities of alleles $A$ and $B$ for each SNP, respectively. Since $M$ is known to be dependent on the overall intensity $S \equiv \log_2(\sqrt{I_AI_B})$,
splines are fit using a mixture model and adjust for this bias using fitted curves. CRLMMv1 then models the distribution of $M$ for a SNP, given the genotype, as Normal. Each SNP is assigned a mean $\mu_i$ and standard deviation $\sigma_i$ for $i = 1, \ldots, I$, which are estimated from the training data. To improve the precision of the model parameters $\mu_i$ and $\sigma_i$, CRLMMv1 uses a hierarchical model using an empirical Bayes approach. They assume the means given the genotype have a multivariate normal distribution and the variances follow an inverse gamma distribution. Then, given the observed log-ratio $M$, the posterior probabilities for each genotype are computed where the estimated parameters are considered known. The posterior probabilities are used as confidence measures. These confidence measures were found to not be optimal and an ad hoc adjustment was proposed, which CRLMMv1 uses. However, there are still considerable limitations to the method, including overly optimistic posteriors, ignoring the statistical uncertainty of estimates from the training step, and failing to model the shift in the genotype parameters from batch to batch.

Carvalho et al. [6] employ an enhanced hierarchical model to address the limitations of CRLMMv1. To estimate the SNP-specific shifts they propose an empirical Bayes approach to estimate the variance of the shifts for each SNP. To estimate the batch-specific shifts they used a two-stage process involving using the previously estimated SNP specific shift parameters to produce preliminary posteriors for each genotype, which were then used to create a pseudo-training dataset. The batch effects associated with each SNP were then estimated using an empirical Bayes approach similar to the one used to estimate the SNP specific shifts. “To account for the uncertainty associated with estimating the SNP- and batch-specific shifts” Carvalho et al. [6] developed a procedure involving producing posterior probabilities. They then produced quality scores for SNPs or batches by assigning a posterior
probability of being an outlier to each shift or batch.

CRLMMv2 assumes

\[
\begin{align*}
Z_{ij} &\ iid\ \text{trinomial}\left( \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right) \\
[\mu_i | Z_{ij} = g] &\ iid\ N_3(0, V) \\
[\lambda_{ij} | \mu_i, Z_{ij} = g] &\ iid\ N_3(0, U_j) \\
[M_{ijk} | \mu_{ig}, \lambda_{ijg}] &= f_{jk}(S_{ijk}) + \mu_{ig} + \lambda_{ijg} + \sigma_{ig}\varepsilon_{ijkg} \\
[\varepsilon_{ijkg} | \mu, \lambda] &\ iid\ t_6(0) \\
\sigma_{ig}^2 &\ iid\ d_g s_g^2 \frac{1}{\chi^2_{dg}}
\end{align*}
\]

where \(i = 1, \ldots, I\) indicates SNP, \(j = 1, \ldots, J\) indicates batch, \(k = 1, \ldots, K\) indicates sample, and \(g = AA, AB, or BB\) are the genotypes. The \(Z_{ij}\) are unobserved, true genotypes, the \(M_{ijk}\) are observed log-ratios, \(\mu_i\) are the shifts for SNP \(i\), \(\lambda_{ij}\) are the batch effects associated with SNP \(i\) and batch \(j\), \(\sigma_{ig}^2\) is the SNP-specific variance for genotype \(g\), \(d_g\) are the degrees of freedom for the variance \(s_g^2\) of a SNP, where \(d_g\) and \(s_g^2\) are estimated from the training data. The hyperparameter \(V\) is estimated using an empirical Bayes approach. The batch-specific shifts \(\lambda\) were estimated using a two-stage process using the previously estimated SNP specific shift parameters. The R/BioConductor package \texttt{crlmm}\ allows implementation of these methods.

Carvalho et al. [6] applied their method to three datasets, two HapMap datasets and one GoKinD dataset from the Genetic Association Information Network (GAIN). They found that their method “accounts for three levels of variability in SNP array data i) SNP-specific shifts, \(ii)\) hybridization batch shifts to each SNP, and \(iii)\) heavy tailed measurement error.”
By explicitly modeling these sources of uncertainty, the estimated posterior probabilities are much improved as compared with those offered by CRLMM version 1.”

1.3.5 Batch Mean Centering (BMC)

Sims et al. [38] employs batch mean-centering (BMC) in their analysis of breast cancer data sets to reduce the variation levels between experiments which allows cross-dataset comparison of the raw transcript levels. Sims et al. [38] found that BMC outperformed distance weighted discrimination (Section 1.3.6) when adjusting for systematic bias in microarray data. Tibshirani et al. [40] include batch mean-centering in their R package pamr that employs their method of nearest shrunken centroid classification to identify subsets of genes that best characterize each class.

BMC simply centers the data within a batch, so that the batch means are all zero. This is also referred to as one-way analysis of variance adjustment by Luo et al. [31]. Mathematically, the batch means across each feature $j$ and within each batch $k$ are calculated as

$$
\bar{x}_{jk} = \frac{1}{n_k} \sum_{i=1}^{n_k} x_{ijk}.
$$

The data $x_{ijk}$ is then adjusted by

$$
x_{ijk}^* = x_{ijk} - \bar{x}_{jk}
$$

and the $x_{ijk}^*$ are used in downstream analyses. Sims et al. [38] found that BMC successfully reduced the amount of between batch (or dataset) variation, while maintaining the within batch variation, allowing multiple batches (or datasets) to be further analyzed together, thus
increasing the statistical power of future analyses.

1.3.6 Distance Weighted Discrimination (DWD)

Marron and Todd [32] present their method Distance Weighted Discrimination (DWD). DWD addresses the generalizability of Support Vector Machines (SVM) and improves upon SVM in high dimension, low sample size (HDLSS) settings. Their new method avoids the problem of “data piling” which is inherent in SVM. DWD computation “is based on computationally intensive optimization, but while the SVM uses well-known quadratic programming algorithms, the DWD uses recently developed interior-point methods for so-called Second-Order Cone Programming (SCOP) problems...The improvement available in HDLSS settings from the DWD comes from solving an optimization problem which yields improved data piling properties.”

Marron and Todd [32] introduce DWD, focusing on two class linear discrimination, meaning that “the discrimination rule is a simple linear function of the new data vector.” They introduce a direction vector $w$ and threshold $\beta$ such that the new data vector $x$ is assigned to the positive class (+1) when $x'w + \beta \geq 0$, where the two classes have labels +1 and −1. Marron and Todd [32] introduce a new optimization method that optimizes the sum of the inverse distances from the data to the separating hyperplane which allows the distances, $r_i$, to influence the direction vector $w$.

Marron and Todd [32] let the training data consist of $n$ vectors $x_i$ of length $p$ with corresponding class indicators $y_i$ where class is indicated by +1 or −1. Then $X$ is the $p \times n$ matrix with columns $x_i$ and $y$ is a length $n$ vector indicating the sample classes. The number of samples in each class can be written $n_+ = \sum_{i=1}^{n} 1_{\{y_i=+1\}}$ and $n_- = \sum_{i=1}^{n} 1_{\{y_i=-1\}}$ so that
\( n = n_+ + n_- \). Marron and Todd [32] let \( Y \) be a \( n \times n \) diagonal matrix with \( y \) on the diagonal. Then, they choose the direction vector (or normal vector) to be \( w \in \mathbb{R}^p \) as the hyperplane and the position to be \( \beta \in \mathbb{R} \). The residual of the \( i \)th data point is then

\[
\tilde{r}_i = y_i(x_i'w + \beta)
\]
or, in matrix notation,

\[
\tilde{r} = Y(X'w + \beta e) = YX'w + \beta y
\]

where \( e \) is a length \( n \) vector of ones. Ideally, \( w \) and \( \beta \) would be chosen such that the residuals are all positive and relatively large. The vector “\( w \) is scaled to have unit norm so that the residuals measure the signed distances of the points from the hyperplane.” Since the positive and negative data might not be able to be separated linearly, an error vector \( \xi \in \mathbb{R}^n_+ \) is added (and penalized) and the perturbed residuals are

\[
r = YX'w + \beta y + \xi
\]

Marron and Todd [32] discuss the optimization problem for the DWD approach in depth. In short, they minimize the sum of the reciprocals of the residuals, perturbed by a penalized vector \( \xi \), such that

\[
\min_{r,w,\beta,\xi} \sum_i (1/r_i) + Ce'\xi, \quad r = YX'w + \beta y + \xi, \quad (1/2)w'w = 1/2, \quad r \geq 0, \quad \xi \geq 0
\]

where \( C > 0 \) is a penalty parameter. They further apply a second-order cone programing (SCOP) problem, an interior-point method, for optimization. Further details on the opti-
mization methods and choice of the tuning parameter $C$ can be found in Marron and Todd [32].

Benito et al. [2] expand on DWD for identifying and adjusting for systematic biases that are present in microarray data sets. They provide methodology to produce batch corrected data matrices using DWD. Their methodology for batch adjustment for binary classes is:

1. Find the DWD direction vector $w$;
2. Project subpopulations in the DWD direction, $(v_+ = x_+ w$ and $v_- = x_- w$);
3. Compute projected subpopulation means ($\mu_+ = \sum v_+/n_+$ and $\mu_- = \sum v_-/n_-)$; and
4. Shift each subpopulation in the DWD direction by an appropriate amount (found by subtracting the DWD direction vector multiplied by each projected mean for each gene; $x_+^* = x_+ - \mu_+ w e$ and $x_-^* = x_- - \mu_- w e$ where $e$ is a length $n$ vector of ones).

This produces a batch corrected data matrix $x^*$ which can be used in further analyses.

Huang et al. [18] further extend binary DWD to the multicategory case and provide a description of their R package R/DWD [17] that implements the classification method distance weighted discrimination (DWD) of Marron and Todd [32] and Benito et al. [2] and their own multiclass method. The batch adjustment procedure of Huang et al. [18] is:

1. Find the $p \times n$ matrix of MDWD direction vectors $w$ which generates a subspace $V$;
2. Project the subpopulations (e.g. respective batch subsets) onto that subspace ($P_{V_k} = X_k w$ where $P_{V_k}$ is $n_k \times b$ for each $k = 1, \ldots, b$);
3. Compute the coordinates of the subpopulation projected means ($\mu_{P_{V_k}} = \frac{1}{n_k} \sum_{i=1}^{n_k} P_{V_k i}$, essentially, the column means of the projection matrix $P_{V_k}$ for each batch $k$); and
4. Shift each subpopulation such that its projected mean is moved in the subspace to a fixed point which is common to all subpopulations ($X_k^* = X_k - \left( w \mu_{P_{V_k}} e_{n_k} \right)'$ where $e_{n_k}$ is a
1 \times n_k \text{ matrix of ones and } \boldsymbol{w} \boldsymbol{\mu}_{\nu_k} \text{ is an } p \times 1 \text{ matrix of the direction matrix multiplied by the projected means for batch } k).

Since the MDWD direction vectors maximize the separation between the batches and ignore the variation in the data, MDWD preserves the variation that is not due to batch effects.

### 1.3.7 Empirical Bayes (EB)

Johnson et al. [22] “propose parametric and non-parametric empirical Bayes frameworks for adjusting data for batch effects that is robust to outliers in small sample sizes and performs comparable to existing methods for large samples.” They apply their method to two microarray data sets. The main benefit of their method over other methods, such as SVD, DWD, and location and scale (L/S) adjustments, is that it works well on small sample sizes where the other methods they mention require more samples per batch since they are not robust to outliers in small sample sizes. Their method estimates the parameters from the L/S model that represent batch effects. This reduces the batch effect parameter estimates to the across genes overall mean of the batch effect estimates by pooling across genes. They then adjust the data for batch effects by using these EB estimates which provides a more robust adjustment for batch effects on each gene. After global normalization, estimation of expression values, and filtering of genes declared absent in more than 80% of samples, the EB method was applied. Johnson et al. [22] show that their method is a very flexible framework for adjusting for additive, multiplicative, and exponential batch effects, and allows for combination of multiple data sets and is robust to small samples sizes.

The EB frameworks assume the data have been globally normalized and thus normalized expression values are available for all features and samples. Let the data contain \(i = 1, \ldots, n\)
samples and \( k = 1, \ldots, b \) batches where each batch includes \( n_k \) samples and \( j = 1, \ldots, p \) features. We assume the model

\[
X_{ijk} = \alpha_j + Y\beta_j + \gamma_{jk} + \delta_{jk}\epsilon_{ijk}
\]

where \( X_{ijk} \) is the normalized expression data, \( \alpha_j \) is the overall expression for feature \( j \), \( Y \) is a design matrix of sample conditions (for example, batch), \( \beta_j \) is the vector of regression coefficients corresponding to \( Y \) for feature \( j \), \( \gamma_{jk} \) are the additive batch effects for batch \( k \) for feature \( j \), and \( \delta_{jk} \) are the multiplicative batch effects for batch \( k \) for feature \( j \). The errors, \( \epsilon_{ijk} \), are assumed to be normally distributed with mean zero and variance \( \sigma^2_j \).

**Step 1: Standardize the data** To avoid the potential for bias due to expression magnitude differences across features, the data are standardized gene-wise to have similar mean and variance as

\[
Z_{ijk} = \frac{X_{ijk} - \hat{\alpha}_j - Y\hat{\beta}_j}{\hat{\sigma}_j}
\]

where the model parameters \( \alpha_j \), \( \beta_j \), and \( \gamma_{jk} \) have been estimated as \( \hat{\alpha}_j \), \( \hat{\beta}_j \), and \( \hat{\gamma}_{jk} \) for \( k = 1, \ldots, b \) and \( j = 1, \ldots, p \). Johnson et al. [22] employ gene-wise ordinary least squares to estimate the parameters and to make sure the parameters are identifiable, they constrain \( \sum_i n_i \hat{\gamma}_{jk} = 0 \) for all \( j = 1, \ldots, p \). The variance can then be estimated as \( \hat{\sigma}^2_j = \frac{1}{N} \sum_{ik} (X_{ijk} - \hat{\alpha}_j - Y\hat{\beta}_j - \hat{\gamma}_{jk})^2 \) where \( N \) is the total number of samples (\( N = \sum_k n_k \)).

**Step 2a: EB batch effect parameter estimates using parametric empirical priors**

The standardized data are assumed to satisfy \( Z_{ijk} \sim N(\gamma_{jk}, \delta^2_{jk}) \). The parametric forms of
the prior distributions of the batch effect parameters are assumed to be

\[ \gamma_{jk} \sim N(X_k, \tau_k^2) \quad \text{and} \quad \delta_{jk}^2 \sim \text{Inverse Gamma}(\lambda_k, \theta_k) \]

where the hyperparameters \( \gamma_k, \tau_k^2, \lambda_k, \) and \( \theta_k \) are estimated empirically using the method of moments from the standardized data. Johnson et al. [22] chose these prior distributions due to their conjugacy with the Normal assumption of the standardized data. Based on the above distributional assumptions, the EB estimates for the batch effects parameters \( \gamma_{jk} \) and \( \delta_{jk}^2 \) are given by the conditional posterior means

\[
\gamma^*_jk = \frac{n_k \tau_k^2 \gamma_{jk} + \delta_{jk}^2 \bar{\gamma}_k}{n_k \tau_k^2 + \delta_{jk}^2} \quad \text{and} \quad \delta^2_{jk} = \frac{\bar{\theta}_k + \frac{1}{2} \sum_i (Z_{ijk} - \gamma^*_jk)^2}{\frac{n_k}{2} + \lambda_k - 1},
\]

respectively.

**Step 2b: EB batch effect parameter estimates using non-parametric empirical priors** The standardized data are assumed to satisfy \( Z_{ijk} \sim N(\gamma_{jk}, \delta_{jk}^2) \) as above. We further assume

\[
\hat{\gamma}_{jk} = \frac{1}{n_k} \sum_i Z_{ijk} \quad \text{and} \quad \hat{\delta}_{jk}^2 = \frac{1}{n_k - 1} \sum_i (Z_{ijk} - \hat{\gamma}_{jk})^2.
\]

The batch effect parameters \( \gamma_{jk} \) and \( \delta_{jk}^2 \) are then estimated using estimates of the posterior expectations of the batch effect parameters, \( E[\gamma_{jk}] \) and \( E[\delta_{jk}^2] \). We let \( Z_{jk} \) be a vector containing \( Z_{ijk} \) for \( i = 1, \ldots, n_k \). Then the posterior expectation of \( \gamma_{jk} \) is

\[
E[\gamma_{jk}] = \int \gamma_{jk} \pi(Z_{jk}, \gamma_{jk}, \delta_{jk}^2) d(\gamma_{jk}, \delta_{jk}^2)
\]

(1.1)
given the posterior distribution $\pi(Z_{jk}, \gamma_{jk}, \delta_{jk}^2)$ of the data $Z_{jk}$ and the batch effect parameters $\gamma_{jk}$ and $\delta_{jk}^2$. Let the unspecified density function for the prior for the parameters $\gamma_{jk}$ and $\delta_{jk}^2$ be $\pi(\gamma_{jk}, \delta_{jk}^2)$ and let the likelihood $L(Z_{jk} | \gamma_{jk}, \delta_{jk}^2) = \prod_i \varphi(Z_{ijk}, \gamma_{jk}, \delta_{jk}^2)$ where $\varphi(Z_{ijk}, \gamma_{jk}, \delta_{jk}^2)$ is the probability density function (pdf) of a random variable distributed $\mathcal{N}(\gamma_{jk}, \delta_{jk}^2)$ and evaluated at $Z_{ijk}$. Equation 1.1 above can then be written

$$E[\gamma_{jk}] = \frac{1}{C(Z_{jk})} \int \gamma_{jk} L(Z_{jk} | \gamma_{jk}, \delta_{jk}^2) \pi(\gamma_{jk}, \delta_{jk}^2) d(\gamma_{jk}, \delta_{jk}^2) \quad (1.2)$$

where $C(Z_{jk}) = \int L(Z_{jk} | \gamma_{jk}, \delta_{jk}^2) \pi(\gamma_{jk}, \delta_{jk}^2) d(\gamma_{jk}, \delta_{jk}^2)$. Johnson et al. [22] then estimated both $C(Z_{jk})$ and the integral in 1.2 using Monte Carlo integration using the empirically estimated $(\gamma_{jk}, \delta_{jk}^2)$ pairs. These pairs are considered random selections from $\pi(\gamma_{jk}, \delta_{jk}^2)$. Finally, if we let $w_{jk''} = L(Z_{jk} | \hat{\gamma}_{jk''}, \hat{\delta}_{jk''}^2)$ for $j'' = 1, \ldots, p$, then we can estimate $C(Z_{jk})$ as $\hat{C}(Z_{jk}) = \frac{1}{n} \sum_{j''} w_{jk''}$ and equation 1.2 can be estimated by

$$\gamma_{jk}^* = \hat{E}[\gamma_{jk}] = \frac{\sum_{j''} w_{jk''} \hat{\gamma}_{jk''}}{n \hat{C}(Z_{jk})}$$

The same method is used to find the posterior expectation of $\delta_{jk}^2$. The non-parametric EB batch adjustments are then given by

$$\gamma_{jk}^* = \frac{\sum_{j''} w_{jk''} \hat{\gamma}_{jk''}}{\sum_{j''} w_{jk''}} \quad \text{and} \quad \delta_{jk}^2 = \frac{\sum_{j''} w_{jk''} \hat{\delta}_{jk''}^2}{\sum_{j''} w_{jk''}}.$$
Step 3: Adjust the data for batch effects The data can now be adjusted using the EB estimated batch effect parameters as

\[ \gamma_{ijk} = \frac{\hat{\sigma}_j}{\hat{\delta}_{jk}}(Z_{ijk} - \hat{\gamma}_{jk}^*) + \hat{\alpha}_j + Y\hat{\beta}_j. \]

Our test statistic \( \delta \) can then be applied to the EB batch corrected data \( \gamma_{ijk}^* \) to test whether EB batch correction successfully corrected the data for batch effects or not.

Konstantinopoulos et al. [24] and Chow et al. [9] apply EB to their datasets for batch correction and Sun et al. [39] applied EB after normalization techniques failed to remove sufficient batch effects from their data. Leek et al. [29] use the EB framework in their R package \texttt{sva} for direct adjustment of known batch effects.

1.3.8 Surrogate Variable Analysis (SVA)
Leek and Storey [27] introduce their method surrogate variable analysis (SVA) to overcome problems caused by heterogeneity in expression studies. They use the term “expression heterogeneity” (EH) to describe patterns of variation due to any un-modeled factor, of which batch effects are one. They consider major sources of expression variation due to technical, environmental, demographic, or genetic factors. They find that applying SVA to data with EH “produces operating characteristics nearly equivalent to what one would obtain with no EH at all.”

SVA allows \( X \) to be the normalized \( p \times n \) expression matrix for \( i = 1, \ldots, n \) arrays and \( j = 1, \ldots, p \) genes and \( y \) to be a length \( n \) vector of the primary variable of interest. Leek and Storey [27] then model \( x_{ij} = \mu_j + f_j(y_i) + e_{ij} \) where \( \mu_j \) is the baseline expression level, \( f_j(y_i) = E(x_{ij}|y_i) - \mu_j \) represents the relationship between the measured variable of interest...
and gene $j$, and $e_{ij}$ is random noise with mean 0. They suppose there are $L$ biologically meaningful unmodeled factors (e.g. age, environmental exposure, genotype, etc.) and $g_l$ ($l = 1, \ldots, L$) is an arbitrarily complicated function of the $l$th factor across all $n$ arrays. The expression of gene $j$ on array $i$ can be modeled by $x_{ij} = \mu_j + f_j(y_i) + \sum_{l=1}^{L} \gamma_{lj} g_{li} + e_{ij}^*$ where $\gamma_{lj}$ is a gene-specific coefficient for the $l$th unmodeled factor and the inter-gene dependent $e_{ij}$ have been replaced by $\sum_{l=1}^{L} \gamma_{lj} g_{li} + e_{ij}^*$ where $e_{ij}^*$ is the true gene-specific noise which is independent across genes and $\sum_{l=1}^{L} \gamma_{lj} g_{li}$ represents dependent variation across genes due to unmodeled factors. Since it is not possible to directly estimate the unmodeled $g_l$, Leek and Storey [27] identify an orthogonal set of vectors $h_k$ for $k = 1, \ldots, K$ and $K \leq L$ with coefficients $\lambda_{ki}$ such that $\sum_{l=1}^{L} \gamma_{lj} g_{li} = \sum_{k=1}^{K} \lambda_{kj} h_{ki}$ and

$$x_{ij} = \mu_j + f_j(y_i) + \sum_{l=1}^{L} \gamma_{lj} g_{li} + e_{ij}^*$$

$$= \mu_j + f_j(y_i) + \sum_{k=1}^{K} \lambda_{kj} h_{ki} + e_{ij}^*$$

The set of $K$ orthogonal vectors $h_k$ are chosen to be the right non-zero singular vectors from the singular value decomposition (SVD) of the $p \times n$ matrix with $(j, i)$ entry $\sum_{l=1}^{L} \gamma_{lj} g_{li}$. The $h_k$ are the surrogate variables. The SVA algorithm then estimates the surrogate variables $h_k$ based on certain consistent expression variation patterns so that they represent signal due to sources other than the primary variable of interest. Leek and Storey [27] provide an algorithm to estimate the surrogate variables. To detect unmodeled factors:

1. “Form estimates $\hat{\mu}_j$ and $\hat{f}_j$ by fitting the model $x_{ij} = \mu_j + f_j(y_i) + e_{ij}$, and calculate the residuals $r_{ij} = x_{ij} - \hat{\mu}_j - \hat{f}_j(y_i)$ to remove the effect of the primary variable on expression. Form the $p \times n$ residual matrix $R$ where the $(j, i)$ element of $R$ is $r_{ij}$."

23
2. Calculate the SVD of the residual expression matrix \( R = UDV' \).

3. Let \( d_l \) be the \( l \)th eigenvalue, which is the \( l \)th diagonal element of \( D \), for \( l = 1, \ldots, n \). If \( df \) is the degrees of freedom of the model fit \( \hat{\mu}_j + \hat{f}_j(y_i) \), then by construction the last \( df \) eigenvalues are exactly zero and we remove them from consideration. For eigengene \( k = 1, \ldots, n - df \) set the observed statistic to be

\[
T_k = \frac{d_k^2}{\sum_{l=1}^{n-df} d_l^2}
\]

which is the variance explained by the \( k \)th eigengene.

4. Form a matrix \( R^* \) by permuting each row of \( R \) independently to remove and structure in the matrix. Denote the \((j,i)\) entry of \( R^* \) by \( r^*_{ij} \).

5. Fit the model \( r^*_ij = \mu^*_j + f^*_j(y_i) + e^*_ij \) and calculate the residuals \( r^0_{ij} = r^*_ij - \hat{\mu}^*_j - \hat{f}^*_j(y_i) \) to form the \( p \times n \) model-subtracted null matrix \( R_0 \).

6. Calculate the SVD of the centered and permuted expression matrix \( R_0 = U_0D_0V'_0 \).

7. For the eigengene \( k \) form a null statistic

\[
T^0_k = \frac{d^2_{0k}}{\sum_{l=1}^{n-df} d^2_{0l}}
\]

as above, where \( d_{0l} \) is the \( l \)th diagonal element of \( D_0 \).

8. Repeat steps 4-7 a total of \( M \) times to obtain null statistics \( T^0_{km} \) for \( m = 1, \ldots, M \) and \( k = 1, \ldots, n - df \).
9. Compute the \( p \)-value for eigengene \( k \) as:

\[
p_k = \frac{\# \{ T_{km}^0 \geq T_k; m = 1, \ldots, M \}}{M}
\]

Since eigengene \( k \) should be significant whenever eigengene \( k' \) is (where \( k' > k \)), we conservatively force monotonicity among the \( p \)-values. Thus, set \( p_k = \max(p_{k-1}, p_k) \) for \( k = 2, \ldots, n - df \).

10. For a user-chosen significance level \( 0 \leq \alpha \leq 1 \), call eigengene \( k \) a significant signature of residual EH if \( p_k < \alpha \).

To construct the surrogate variables:

1. “Form estimates \( \hat{\mu}_j \) and \( \hat{f}_j \) by fitting the model \( x_{ij} = \mu_j + f_j(y_i) + e_{ij} \), and calculate the residuals \( r_{ij} = x_{ij} - \hat{\mu}_j - \hat{f}_j(y_i) \) to remove the effect of the primary variable on expression. Form the \( p \times n \) residual matrix \( R \) where the \((j, i)\) element of \( R \) is \( r_{ij} \).

2. Calculate the SVD of the residual expression matrix \( R = UDV' \). Let \( e_k = (e_{k1}, \ldots, e_{kn})' \) be the \( k \)th column of \( V \) (for \( k = 1, \ldots, n \)). These \( e_k \) are the residual eigengenes and represent orthogonal residual EH signals independent of the signal due to the primary variable.

3. Set \( \hat{K} \) to the number of significant eigengenes found by the above algorithm. Note that “significant” means that the eigengene represents a greater proportion of variation than expected by chance.

4. For each significant eigengene \( e_k, k = 1, \ldots, \hat{K} \), regress \( e_k \) on the \( x_j \) \((j = 1, \ldots, p)\) and calculate a \( p \)-value testing for an association between the residual eigengene and
each gene’s expression. This $p$-value measures the strength of association between the residual eigengene $e_k$ and the expression for gene $j$.

5. Let $\pi_0$ be the proportion of genes with expression not truly associated with $e_k$; form an estimate $\hat{\pi}_0$ and estimate the number of genes associated with the residual eigengene by $\hat{p}_1 = [(1 - \hat{\pi}_0 \times p)]$. Let $s_1, \ldots, s_{\hat{p}_1}$ be the indices of the genes with $\hat{p}_1$ smallest $p$-values from this test.

6. Form the $\hat{p}_1 \times n$ reduced expression matrix $X_r = (x_{s_1}, \ldots, x_{s_{\hat{p}_1}})'$. Since $\hat{m}_1$ is an estimate of the number of genes associated with residual eigengene $k$, the reduced expression matrix represents the expression of those genes estimated to contain the EH signature represented by so $h_k$ as described above. As was done for $R$, calculated the eigengenes of $X_r$, and denote these by $e_i^r$ for $i = 1, \ldots, n$.

7. Let $i^* = \arg \max_{1 \leq i \leq n} \langle e_k, e_i^r \rangle$ and set $\hat{h}_k = e_{i^*}^r$. In other words, set the estimate of the surrogate variable to be the eigengene of the reduced matrix most correlated with the corresponding residual eigengene. Since the reduced matrix is enriched for genes associated with this residual eigengene, this is a principled choice for the estimated surrogate variable that allows for correlation with the primary variable.

8. In any subsequent analysis, employ the model $x_{ij} = \mu_j + f_j(y_i) + \sum_{k=1}^{K} \lambda_{kj} h_{ki} + e_{ij}^*$, which serves as an estimate of the ideal model $x_{ij} = \mu_j + f_j(y_i) + \sum_{k=1}^{K} \lambda_{kj} h_{ki} + e_{ij}^*$.

Leek et al. [29] discuss their package sva for identifying, estimating, and removing batch effects in high-throughput experiments. Their package uses surrogate variable estimation with the sva function, direct adjustment for known batch effects with the Combat function [22], and adjustment for batch and latent variables in prediction problems with the fsva
function (similar to fRMA of McCall et al. [33]).

1.4 Evaluation Method

Various methods exist to evaluate the many different batch effect adjustment methods. Lazar et al. [26] discuss both qualitative and quantitative methods. To evaluate batch effect removal tools visually they recommend boxplots of gene expression data of two or more experiments to be combined, density plots that show the distribution of gene expression values for a few randomly selected genes, dendrograms resulting from hierarchical clustering analyses (as previously discussed in 1.2.1), principal components plots (also previously discussed in 1.2.2), and relative log expression plots (a boxplot for each sample of the deviation of the median log expression for each gene from the sample median log expression value). Lazar et al. [26] discuss several quantitative measures of evaluation including principal variance component analysis (PVCA) and correlation coefficients. PVCA combines two data analysis methods, principal components analysis and variance components analysis, and is used as a method to determine the sources of variation in data [35]. Lazar et al. [26] ultimately categorize the evaluated methods based on model complexity, the minimum number of samples required, the number of datasets required, covariate flexibility, requirement of additional prior information, and computational time. See [26] for more detail on these methods.

Chen et al. [8] assess the amount of variation due to batch before and after batch adjustment using PVCA, the precision of the batch adjustment method using correlation among replicates (either Pearson’s correlation for pairs or intraclass correlation for groups), the accuracy of the batch adjustment method using the correlation between nominal fold change and observed fold change, and the overall batch effect adjustment performance using ROC
curves and area under the curve (AUC). They compared DWD, mean-centering (BMC), SVA, geometric ratio-based method (Ratio_G), and EB using both parametric (ComBat_p) and non-parametric (ComBat_n) methods. By classifying these six batch adjustment methods using the above evaluation methods, Chen et al. [8] were able to assess the precision, accuracy, and overall performance of each of them and give a better comparison of the true abilities of each batch adjustment method. They ultimately found that either EB method outperformed the other batch adjustment methods.

1.5 Guided Principal Component Analysis (gPCA)

We propose a test statistic derived using both the traditional PCA method and a new method, guided PCA (gPCA; see Chapter 2) [34], for detecting batch effects. We evaluate the performance of our test in extensive simulation studies (Chapter 3). We also demonstrate the difference between PCA and gPCA using two copy number variation datasets (Chapter 4). Though our illustration pertains to copy number data, the methods are appropriate for any type of high-throughput genomic data. Our proposed test statistic may be useful for identifying whether any of the listed batch adjustment methods should be applied prior to statistical analysis. The effectiveness of different batch effect adjustment methods is assessed by applying our test statistic to the raw data then subsequently to batch corrected data (Chapter 5). An R program that implements our test statistic is described (Chapter 6). Conclusions and future directions are discussed in Chapter 7.
2 Statistical Methods

2.1 Principal Components Analysis (PCA)
Principal components analysis (PCA) is used for data reduction and interpretation. It is used to explain the variance-covariance structure of a set of variables through linear combinations of the variables [21]. PCA is a form of unsupervised learning that seeks to find the “combination of conditions that explain the greatest variation in the data”[41]. It is used in many types of analyses including neuroscience and computer graphics [37], in addition to microarray data analyses [16, 41]. The numerical workhorse of PCA is singular-value decomposition (SVD).

Singular-value decomposition (SVD) Let $X$ be a centered $n \times p$ matrix of real numbers where $n$ denotes sample and $p$ denotes genomic feature (e.g., probe). Then there exists an $n \times n$ orthogonal matrix $U$ and a $p \times p$ orthogonal matrix $V$ such that

$$X = UDV'$$

where the $n \times p$ matrix $D$ has diagonal $(q, q)$ entry $\lambda_q \geq 0$ for $q = 1, \ldots, \min(n, p)$ where, by convention, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{\min(n,p)}$ and the other entries are zero. The positive constants
\( \lambda_q \) are called the *singular values* of \( D \) [21]. The matrix \( D \) has form

\[
\begin{bmatrix}
\lambda_1 & 0 & \cdots & 0 \\
0 & \lambda_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \lambda_n \\
\end{bmatrix}
\]

when we let \( \min(n, p) = n \).

Principal components are the length \( n \) column vectors \((P_1, P_2, \ldots, P_p)\) of

\[
P = X \mathbf{V}
\]

where \( X \) is an \( n \times p \) matrix of features, \( \mathbf{V} \) is the \( p \times p \) matrix of right singular vectors, \( \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p \), from the singular value decomposition and \( P \) is the \( n \times p \) principal components matrix.

The first principal component is the linear combination of the variables having the largest variance, the second principal component has the next largest variance under the constraint that it is uncorrelated with the proceeding principal component, etc. Typically, PCA is performed on \( X \) alone after standardizing each variable in \( X \). Herein, we refer to this as “unguided” PCA. Unguided PCA finds a linear combination (or projection) of variables in
X with coefficients from V with maximum variance. As discussed in Chapter 1, use of unguided PCA is subjective as one is required to interpret a plot of the first and second principal components against one another. Moreover, unguided PCA is not effective for identifying batch effects if they are not the largest source of variation. In this case, it does not mean that batch effects do not exist in the data, but that alternate methods must be used to find them.

2.2 Guided PCA

For detecting batch effects, a more informative version of PCA is on $Y'X$ where $Y$ is an $n \times b$ indicator matrix where $b$ denotes batch. Each batch is comprised of $n_k$ observations such that $\sum_k n_k = n$. The indicator matrix consists of $b$ blocks with $n_k$ rows, $k = 1, \ldots, b$, and $k$ columns where, for each block,

$$Y_b = \begin{cases} 
1 & k=b \\
0 & \text{otherwise}
\end{cases}$$

so that

$$Y = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 
\end{bmatrix}$$

where $1$ and $0$ are vectors with

$$y_{ik} = \begin{cases} 
1 & \text{if sample } i \text{ is in batch } k \\
0 & \text{otherwise}
\end{cases}$$
for $i = 1, \ldots, n$ and $k = 1, \ldots, b$. Performing SVD on $Y'X$ results in a $b \times b$ matrix $U$ that denotes the batch loadings and the $p \times p$ matrix $V$ that denotes the probe loadings. Large singular values imply that the batch is important for the corresponding principal component. gPCA guides the SVD to look for batch effects in the data based on the batch indicator matrix $Y$, which can be defined to indicate any type of potential batch effect.

Another commonly used method in this situation is Canonical Correlation Analysis (CCA), which finds the linear combination with maximum correlation; however, we are interested in variance, not correlation.

### 2.3 Proposed Method: Test statistic for testing if batch effect exists

Our test statistic, $\delta$, quantifies the proportion of variance due to batch effects in experimental genomic data. The proportion of total variance due to batch is taken to be the ratio of the variance of the first principal component from gPCA to the variance of the first principal component from unguided PCA

$$\delta = \frac{\text{var}(XV_g)}{\text{var}(XV_u)}$$

where $g$ indicates gPCA and $u$ indicates unguided PCA. $V$ is the matrix of probe loadings resulting from gPCA or PCA, respectively. Large values of $\delta$ (values near 1) imply that the batch effect is large.

To determine whether $\delta$ is significantly larger than would be expected by chance, a $p$-value is estimated using a permutation distribution created by permuting the batch vector $M = 1000$ times so that $\delta_{pm}$ is computed for $m = 1, \ldots, M$ where $p$ indicates the permutation. Here $\delta_{pm}$ is the proportion of the total variance due to the first principal component from the $m^{th}$ permutation from gPCA to the total variance due to the first principal component from unguided PCA. A one-sided $p$-value (testing $H_0 : \delta_{pm} = \delta$ versus $H_1 : \delta_{pm} > \delta$) is estimated as
the proportion of times the observed $\delta$ was in the extreme tail of the permutation distribution

$$p\text{-value} = \frac{\sum_{m=1}^{M} (\delta_{p_m} > \delta)}{M}.$$  

**Estimating percent of total variation explained by batch** The percent of total variation explained by batch is then calculated as

$$\frac{\hat{PC}_g - \hat{PC}_u}{\hat{PC}_g} \times 100$$

where

$$\hat{PC}_u = \frac{\text{var}(XV_{u1})}{\sum_{i=1}^{n} \text{var}(XV_{ui})} \quad \text{and} \quad \hat{PC}_g = \frac{\text{var}(XV_{g1})}{\sum_{k=1}^{b} \text{var}(XV_{gk})}$$

where $u$ and $g$ represent unguided PCA and gPCA, respectively.

All analyses were performed in R 2.15.2.
3 Simulation Study

3.1 Description of Simulation Study

Most often investigators are interested in modeling their data in the presence of a known phenotype. Therefore, we simulated data to represent copy number data under three scenarios: (1) feature data (here, feature denotes probe) with no phenotypic variable; (2) feature data having a high variance phenotypic effect; and (3) feature data having a low variance phenotypic effect. The feature data were generated from an independent normal distribution with \( p = 1000 \) features and \( n = 90 \) observations. To study Type I and II errors, for all three scenarios, the data were simulated in two ways, to include a true batch effect and without a true batch effect.

3.1.1 Evaluating Type I Error

To evaluate Type I error for all three scenarios, the data were simulated without a true batch effect. The resulting proportion of \( p \)-values that were < 0.05 formed our estimate of the Type I error. Type I error was estimated when the variance associated with batch was \( \sigma_b = 0.5 \) and 1 for all three scenarios. For the low variance phenotype scenario, Type I error was additionally assessed when the proportion of features affected by the phenotype was 0.1 or 0.05. Table 3.1 shows the parameters for all Type I error simulation scenarios.

3.1.2 Evaluating Power

To study power for all three scenarios, the data were simulated with a true batch effect. When a batch effect was present, there were two batches with means \( \mu_{b_1} = 0 \) and \( \mu_{b_2} = 1 \) and an equal number of observations in each batch. We varied the variance within each batch allowing it to be either \( \sigma_b = 0.5 \) or \( \sigma_b = 1 \) for each feature and features were generated
Table 3.1: Type I Error Simulation Scenarios: \( n = 90 \) and \( p = 1000 \) for all scenarios. \( \mu_{b_1} \) and \( \mu_{b_2} \) are the batch means, \( \sigma_b \) is the variance associated with batch, \( \mu_{p_1} \) and \( \mu_{p_2} \) are the phenotypic means, \( \sigma_p \) is the variance associated with phenotype, and \( p_{\text{prop}} \) is the proportion of features effected by phenotype.

<table>
<thead>
<tr>
<th></th>
<th>( \mu_{b_1} )</th>
<th>( \mu_{b_2} )</th>
<th>( \sigma_b )</th>
<th>( \mu_{p_1} )</th>
<th>( \mu_{p_2} )</th>
<th>( \sigma_p )</th>
<th>( p_{\text{prop}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Phenotype</td>
<td>(-)</td>
<td>(-)</td>
<td>0.5 or 1.0</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td>High Variance Phenotype</td>
<td>0</td>
<td>1</td>
<td>0.5 or 1.0</td>
<td>0</td>
<td>1</td>
<td>2.0</td>
<td>0.1</td>
</tr>
<tr>
<td>Low Variance Phenotype</td>
<td>0</td>
<td>1</td>
<td>0.5 or 1.0</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
<td>0.1 or 0.05</td>
</tr>
</tbody>
</table>

Independently. The features with a batch effect were simulated from multivariate normal distribution such that

\[
\begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
\sim N_{n_1}(\mu_{b_1}, \sigma_b^2 I)
\quad \text{and} \quad
\begin{bmatrix}
  b_2
\end{bmatrix}
\sim N_{n_2}\left(\begin{bmatrix}
  \mu_{b_1} \\
  \mu_{b_2}
\end{bmatrix}, \sigma_b^2 I\right)
\]

where \( b_2 \) has mean vector \( \mu_{b_2} \) of length \( b_{\text{prop}} \times p \) for a proportion of the features specified by \( b_{\text{prop}} \) and mean vector \( \mu_{b_1} \) of length \((1 - b_{\text{prop}}) \times p \) for a proportion of the features specified by \( 1 - b_{\text{prop}} \). The batch feature data were randomly assigned to observations based on the randomly assigned batch labels. For all three scenarios, the proportion of \( p \)-values \( < 0.05 \) formed our estimate of the power. Power was estimated at varying levels of the proportion of features affected by batch. Table 3.2 shows the parameters for all power simulation scenarios.

### 3.1.3 Simulating Phenotypic Effects

In the true phenotype scenarios, 10\% of the features were affected by phenotype using means \( \mu_{p_1} = 0 \) and \( \mu_{p_2} = 1 \) and variance \( \sigma_p = 2 \) for the high variance scenario and \( \sigma_p = 0.2 \) for the low variance scenario. The proportion of features affected by the phenotype (\( p_{\text{prop}} \)) was 0.1 or 0.05. The phenotypic effect was simulated from a multivariate normal distribution such
Table 3.2: Power Simulation Scenarios: \( n = 90 \) and \( p = 1000 \) for all scenarios. \( b \) denotes the number of batches, \( n_1 \) and \( n_2 \) give the number of samples in each batch, \( \mu_{b_1} \) and \( \mu_{b_2} \) are the batch means, \( \sigma_b \) is the variance associated with batch, \( \mu_{p_1} \) and \( \mu_{p_2} \) are the phenotypic means, \( \sigma_p \) is the variance associated with phenotype, and \( p_{\text{prop}} \) is the proportion of features effected by phenotype.

<table>
<thead>
<tr>
<th></th>
<th>( b )</th>
<th>( n_1 )</th>
<th>( n_2 )</th>
<th>( \mu_{b_1} )</th>
<th>( \mu_{b_2} )</th>
<th>( \sigma_b )</th>
<th>( \mu_{p_1} )</th>
<th>( \mu_{p_2} )</th>
<th>( \sigma_p )</th>
<th>( p_{\text{prop}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Phenotype</td>
<td>2</td>
<td>45</td>
<td>45</td>
<td>0</td>
<td>1</td>
<td>0.5 or 1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>High Variance Phenotype</td>
<td>2</td>
<td>45</td>
<td>45</td>
<td>0</td>
<td>1</td>
<td>0.5 or 1.0</td>
<td>0</td>
<td>1</td>
<td>2.0</td>
<td>0.1</td>
</tr>
<tr>
<td>Low Variance Phenotype</td>
<td>2</td>
<td>45</td>
<td>45</td>
<td>0</td>
<td>1</td>
<td>0.5 or 1.0</td>
<td>0</td>
<td>1</td>
<td>0.2</td>
<td>0.1 or 0.05</td>
</tr>
</tbody>
</table>

that

\[
\begin{bmatrix}
  p_1 \\
p_2
\end{bmatrix}
\]

where \( p_1 \sim N_{n_1}(\mu_{p_1}, \sigma^2_{p_1}I) \) and \( p_2 \sim N_{n_2}
\begin{bmatrix}
  \mu_{p_1} \\
  \mu_{p_2}
\end{bmatrix}
\begin{bmatrix}
  \sigma^2_{p_1}I
\end{bmatrix}
\]

where \( p_2 \) has mean vector \( \mu_{p_2} \) of length \( p_{\text{prop}} \cdot p \) for a proportion of the features specified by \( p_{\text{prop}} \) and mean vector \( \mu_{p_1} \) of length \( (1 - p_{\text{prop}}) \cdot p \) for a proportion of the features specified by \( 1 - p_{\text{prop}} \). In all scenarios with a phenotypic effect, the phenotype was generated independent from batch effect.

An additional phenotypic simulation was performed that allowed phenotype and batch to be dependent. In this scenario each feature \( j \) for \( j = 1, \ldots, p \) was assigned to have no phenotypic effect, a phenotypic effect only, a batch effect only, or both batch and phenotypic effects. For feature \( j \), we let

\[
f_j = \beta_{p}p_{j}\text{pheno} + \beta_{b}b_{j}\text{batch} + e
\]

where \( p \) and \( b \) are length \( p \) vectors indicating whether each feature had a phenotypic or batch
effect, respectively, **pheno** and **batch** are length $n$ vectors giving the phenotype and batch effect for each sample, and $e \sim N(0, \sigma_b)$ is a random error term. The $\beta_p$ and $\beta_b$ parameters determine the magnitude of the phenotypic and batch effects, respectively. If feature $j$ has both a phenotype effect and a batch effect then $p = b = 1$, if $j$ has only a phenotype effect then $p = 1$ and $b = 0$, if $j$ has only a batch effect then $p = 0$ and $b = 1$, and finally if $j$ has neither effect then $p = b = 0$. The $f_j$ feature vectors for $j = 1, \ldots, p$ form our $n \times p$ feature data matrix $X$. The vectors $b$ and $p$ can be calculated in two ways. Either as in the independent continuous simulation, where a random sample of the $p$ features determined by $b_{\text{prop}}$ and $p_{\text{prop}}$, respectively, and since they are no longer independent any overlap in the features determines which features have both effects. Alternatively, the number of features with each effect can be specifically set so that the exact number in each group, batch effect, phenotype effect, and batch and phenotype effect, is specified, so the number with both effects is predetermined. For our dependent continuous phenotype simulation, the simulation parameters were $\beta_p = 0.5$ and $\beta_b = 1$ and otherwise as in Table 3.2, however, for this scenario, the batch and phenotype means are unnecessary so were not used.

Each simulation scenario was repeated 500 times. Phenotype here can be thought of as any variable of interest, whether categorical (e.g., case versus control, smoker versus non-smoker) or continuous (e.g., mammographic density, age, body mass index).

### 3.2 Results

The estimates for Type I error for all scenarios are reported in Table 3.3. The proportion of features with a phenotypic effect is $p_{\text{prop}} = 0.1$ for scenarios (b-c) and 0.05 for scenario (d). In all scenarios, the Type I error is at or below the nominal 0.05 level. Figure 3.1 shows power of our test statistic as a function of the proportion of features affected by batch
if there is no true phenotypic effect. If \( \sigma_b = 0.5 \), then our test statistic has 80% power if approximately 0.3% of the features are affected by batch. If \( \sigma_b = 1 \), then approximately 0.5% of features need to have a batch effect in order to achieve 80% power. If a high variance phenotypic effect exists, then approximately 1.5% or 2% of the features need to have a batch effect in order to achieve 80% power for \( \sigma_b = 0.5 \) and \( \sigma_b = 1 \), respectively (Figure 3.2). Similarly, if a low variance phenotype exists and 10% of features are affected by phenotype, then approximately 1.5% or 2% of the features need to have a batch effect in order to achieve 80% power for \( \sigma_b = 0.5 \) and \( \sigma_b = 1 \), respectively, and if 5% of features are affected by phenotype, then approximately 0.75% of the features need to have a batch effect in order to achieve 80% power for both \( \sigma_b = 0.5 \) and \( \sigma_b = 1 \) (Figure 3.3). Therefore, if a phenotypic effect is truly present, a larger proportion of features need to be affected by batch in order to detect if a batch effect is present compared to when there is no phenotypic effect present in the feature data. Figure 3.4 shows the cumulative variance of the unguided and guided principal components for the low variance phenotypic simulated data which indicates that when the batch variance is larger, the proportion of the variance explained by the first principal component is smaller than when the true batch variance is smaller.

Power is also higher when the batch variance is smaller given the same level of separation in batch means. Further simulations with batch and phenotype effects simulated so that they are not independent, varying the batch variance, with the difference between batch means smaller than the difference between the phenotype means, and with high proportions of features affected by batch can be found in Sections 3.2.1, 3.2.2 and 3.2.3. Additionally, data for the three scenarios were simulated and a sensitivity to batch correction analysis was performed using the batch mean centering method of Sims et al. [38] (Section 5.4.1).
Table 3.3: Estimated Type I Error: For all scenarios there is no true batch effect. Scenario (a) has no phenotypic effect in the data, however scenario (b) has a high variance phenotypic effect included in the analysis with phenotypic effect at $p_{prop}=0.1$ and scenarios (c-d) have low variance phenotypic effects included in the analysis with phenotypic effect at $p_{prop}=0.1$ or 0.05, respectively.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$\sigma_b = 0.5$</th>
<th>$\sigma_b = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) No Phenotype</td>
<td>0.034</td>
<td>0.034</td>
</tr>
<tr>
<td>(b) High Variance Phenotype ($p_{prop}=0.1$)</td>
<td>0.014</td>
<td>0.014</td>
</tr>
<tr>
<td>(c) Low Variance Phenotype ($p_{prop}=0.1$)</td>
<td>0.000</td>
<td>0.002</td>
</tr>
<tr>
<td>(d) Low Variance Phenotype ($p_{prop}=0.05$)</td>
<td>0.010</td>
<td>0.046</td>
</tr>
</tbody>
</table>

3.2.1 Dependent Batch and Phenotype Effects

The true phenotype simulations with batch and phenotype as independent does not take into account the possibility that a feature will have both a batch and phenotype affecting it. We performed an additional simulation that simulated the feature data such that some features had both a batch and phenotype effect. Figure 3.5 shows power for our test statistic when batch effect and phenotype are simulated so that they are not independent. Between 0.2% and 0.3% of features need to have a batch effect in order to achieve 80% power when $\sigma = 0.5$ and approximately 0.55% of features need to have batch effect in order to achieve 80% power when $\sigma = 1$. In this scenario, similar to the above simulations, power is higher when variance is smaller.

3.2.2 Varied Batch Variance and Phenotypic Means Greater than Batch Means

The sensitivity of gPCA results to the level of batch variance was assessed through additional simulation analyses. For the no phenotype, high variance phenotype, and low variance phenotype scenarios, estimated power was calculated while varying the variance associated with batch between $\sigma_b = 0.5$ and $\sigma_b = 2$. The proportion of features affected by batch
Figure 3.1: Power for detecting batch effect as a function of the proportion of features that are affected by batch when no true phenotype was included with batch proportion ranging from 0.1 to 1%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$.

(bprop) were held constant at 0.010, 0.050, and 0.100 for the no phenotype, high variance phenotype, and low variance phenotype scenarios, respectively. The batch means were held constant at $\mu_{b_1} = 0$ and $\mu_{b_2} = 1$. These values of the batch proportion and means were found to have good power when varying the batch proportion in our previous simulations. For the true phenotype scenarios, the phenotype means were also varied as an assessment of gPCA
Figure 3.2: Power for detecting batch effect as a function of the proportion of features that are affected by batch when a true high variance phenotype was included in the data with batch proportion ranging from 0.1 to 2.5%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$.

![Power plots for three scenarios](image)

when the phenotypic means ($\mu_{p1} = 0$ and $\mu_{p2} = 1.5$ or $\mu_{p2} = 2$) are higher than the batch means ($\mu_b = 0$ and $\mu_b = 1$). Figure 3.6 shows the power plots for the three scenarios.

We found that as batch variance increased, so did the estimated power and the smaller the difference in the phenotypic means, the higher the power. In the no phenotype scenario,
Figure 3.3: Power for detecting batch effect as a function of the proportion of features that are affected by batch when a true low variance phenotype was included in the data with batch proportion ranging from 0.1 to 2.5%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$ and the proportion of features affected by phenotype is either $\text{pprop} = 0.1$ or $\text{pprop} = 0.05$.

when holding the batch means fixed, we found that power decreased as the batch variance increased since when there is no phenotype, the unguided PCA has only the batch variance as does gPCA.
3.2.3 High Proportion of Features Affected by Batch

It is of interest to investigate the performance of gPCA when the proportion of features affected by batch is high. Simulations were assessed with batch proportion between 50 and 90% of features. Table 3.4 shows the estimated power is 100% for all scenarios so good results can be expected even when a large proportion of features are affected by batch.

3.2.4 Sensitivity of gPCA Results to Filtering of Simulation Data

Data were also simulated as in our main simulation study, but with \( p = 20,000 \) features and \( n = 90 \) samples. For each of the three phenotype scenarios, no phenotype, high variance phenotype, and low variance phenotype, data were simulated with batch and phenotype means \( \mu_{b_1} = \mu_{p_1} = 0 \) and \( \mu_{b_2} = \mu_{p_2} = 1 \) where \( p \) denotes phenotype and \( b \) denotes batch.
Table 3.4: Power for detecting batch effect as a function of the proportion of features that are affected by batch at 50 to 90% when no phenotypic, high variance phenotypic, or low variance phenotypic data were included in gPCA.

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>bprop</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.500</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.633</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.767</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.900</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.500</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.633</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.767</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.900</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) No Phenotype

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>$\sigma_p$</th>
<th>bprop</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>2.0</td>
<td>0.500</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>2.0</td>
<td>0.633</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>2.0</td>
<td>0.767</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>2.0</td>
<td>0.900</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) High Variance Phenotype

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>$\sigma_p$</th>
<th>pprop</th>
<th>bprop</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.05</td>
<td>0.500</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.05</td>
<td>0.633</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.05</td>
<td>0.767</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.05</td>
<td>0.900</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.10</td>
<td>0.500</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.10</td>
<td>0.633</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.10</td>
<td>0.767</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.10</td>
<td>0.900</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.05</td>
<td>0.500</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.05</td>
<td>0.633</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.05</td>
<td>0.767</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.05</td>
<td>0.900</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.10</td>
<td>0.500</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.10</td>
<td>0.633</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.10</td>
<td>0.767</td>
<td>1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.10</td>
<td>0.900</td>
<td>1</td>
</tr>
</tbody>
</table>

(c) Low Variance Phenotype
Figure 3.5: Power for detecting batch effect as a function of the proportion of features that are affected by batch when a true phenotype was included in the data and batch and phenotype effect are not independent. The batch proportion ranges from 0.1 to 1%. The variance associated with batch is $\sigma_b = 0.5$ or $\sigma_b = 1.0$ and the proportion of features affected by phenotype is either $p_{prop} = 0.1$ or $p_{prop} = 0.05$. The parameters $\beta_p = 0.5$ and $\beta_b = 1$.

The batch variance was $\sigma_b = 0.5$, the phenotype variance for the high variance scenario was $\sigma_p = 2$, and the phenotype variance for the low variance scenario was $\sigma_p = 0.2$. The proportion of features affected by batch was held constant at 0.004, 0.017, and 0.025 for the
no, high variance, and low variance phenotype scenarios, respectively, each of which had good power in the previous simulation study. The proportion of features affected by phenotype in the high variance and low variance scenarios was $p_{\text{prop}} = 0.01$. Table 3.5 shows the resulting $p$-values from retaining between 10 and all features from the simulated data sets. The test statistic applied to the simulated data were not affected by filtering provided that the percent of features retained was 5% (1000 features) when there was a phenotype with high variance (a somewhat weak phenotypic effect) and approximately 50% (10,000 features) when there was a phenotype with low variance (that is, a strong phenotypic effect) and thus filtering can be used as a method to reduce the analysis time required provided it is judiciously applied (Table 3.5).

### 3.2.5 Analysis of Varying Batch Sample Size

A simulation analysis was performed that varied the batch sample sizes allowing the sample sizes of the two batches to be vary between $n_k = 12$ and 84 for $k = 1, \ldots, b$ where $b = 2$. Table 3.6 shows the results of these analyses. There is no difference in either the value of $\delta$ or the $p$-values when computing the batch indicator matrix $Y$ with 0’s and 1’s indicating whether a sample is in batch $k$ or not versus computing $Y$ scaling by batch sample size $n_k$ (i.e. with 0’s and $1/n_k$’s). These results indicate that scaling the batch indicator matrix by batch sample size $n_k$ has no effect on our gPCA $\delta$ test statistic results.
Table 3.5: Variance Filtering Sensitivity Results: $\delta$, corresponding $p$-values resulting from retaining between 10 and all features from the simulation data sets. The last column gives the system time in minutes required to run gPCA as discussed in Section 4.2.4.

<table>
<thead>
<tr>
<th>Features</th>
<th>% Features</th>
<th>$\delta$</th>
<th>$p$-value</th>
<th>System Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.05</td>
<td>0.9991</td>
<td>&lt; 0.001</td>
<td>0.027</td>
</tr>
<tr>
<td>100</td>
<td>0.50</td>
<td>0.9976</td>
<td>&lt; 0.001</td>
<td>0.030</td>
</tr>
<tr>
<td>1000</td>
<td>5.00</td>
<td>0.9918</td>
<td>&lt; 0.001</td>
<td>0.048</td>
</tr>
<tr>
<td>2000</td>
<td>10.00</td>
<td>0.9896</td>
<td>&lt; 0.001</td>
<td>0.069</td>
</tr>
<tr>
<td>5000</td>
<td>25.00</td>
<td>0.9856</td>
<td>&lt; 0.001</td>
<td>0.133</td>
</tr>
<tr>
<td>10000</td>
<td>50.00</td>
<td>0.9839</td>
<td>&lt; 0.001</td>
<td>0.241</td>
</tr>
<tr>
<td>15000</td>
<td>75.00</td>
<td>0.9787</td>
<td>&lt; 0.001</td>
<td>0.350</td>
</tr>
<tr>
<td>20000</td>
<td>100.00</td>
<td>0.9795</td>
<td>&lt; 0.001</td>
<td>0.445</td>
</tr>
</tbody>
</table>

(a) No Phenotype

<table>
<thead>
<tr>
<th>Features</th>
<th>% Features</th>
<th>$\delta$</th>
<th>$p$-value</th>
<th>System Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.05</td>
<td>0.5571</td>
<td>0.673</td>
<td>0.029</td>
</tr>
<tr>
<td>100</td>
<td>0.50</td>
<td>0.4314</td>
<td>0.086</td>
<td>0.029</td>
</tr>
<tr>
<td>1000</td>
<td>5.00</td>
<td>0.4015</td>
<td>0.037</td>
<td>0.049</td>
</tr>
<tr>
<td>2000</td>
<td>10.00</td>
<td>0.4429</td>
<td>0.021</td>
<td>0.071</td>
</tr>
<tr>
<td>5000</td>
<td>25.00</td>
<td>0.4977</td>
<td>0.009</td>
<td>0.138</td>
</tr>
<tr>
<td>10000</td>
<td>50.00</td>
<td>0.5495</td>
<td>0.004</td>
<td>0.252</td>
</tr>
<tr>
<td>15000</td>
<td>75.00</td>
<td>0.5846</td>
<td>0.003</td>
<td>0.366</td>
</tr>
<tr>
<td>20000</td>
<td>100.00</td>
<td>0.6211</td>
<td>0.002</td>
<td>0.469</td>
</tr>
</tbody>
</table>

(b) High Variance Phenotype

<table>
<thead>
<tr>
<th>Features</th>
<th>% Features</th>
<th>$\delta$</th>
<th>$p$-value</th>
<th>System Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.05</td>
<td>0.9968</td>
<td>&lt; 0.001</td>
<td>0.026</td>
</tr>
<tr>
<td>100</td>
<td>0.50</td>
<td>0.9996</td>
<td>&lt; 0.001</td>
<td>0.028</td>
</tr>
<tr>
<td>1000</td>
<td>5.00</td>
<td>0.9998</td>
<td>&lt; 0.001</td>
<td>0.047</td>
</tr>
<tr>
<td>2000</td>
<td>10.00</td>
<td>0.9996</td>
<td>&lt; 0.001</td>
<td>0.067</td>
</tr>
<tr>
<td>5000</td>
<td>25.00</td>
<td>0.9993</td>
<td>&lt; 0.001</td>
<td>0.130</td>
</tr>
<tr>
<td>10000</td>
<td>50.00</td>
<td>0.9988</td>
<td>&lt; 0.001</td>
<td>0.238</td>
</tr>
<tr>
<td>15000</td>
<td>75.00</td>
<td>0.9985</td>
<td>&lt; 0.001</td>
<td>0.351</td>
</tr>
<tr>
<td>20000</td>
<td>100.00</td>
<td>0.3913</td>
<td>0.027</td>
<td>0.452</td>
</tr>
</tbody>
</table>

(c) Low Variance Phenotype
Table 3.6: Test statistic $\delta$ and corresponding $p$-value resulting from varying the batch sample sizes between 12 and 84. $\delta_s$ and $p_s$ indicate the results of the test when considering the batch effect indicator matrix $Y$ as scaled by batch sample size $n_k$.

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$\delta$</th>
<th>$p$</th>
<th>$\delta_s$</th>
<th>$p_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>0.827</td>
<td>0.257</td>
<td>0.827</td>
<td>0.257</td>
</tr>
<tr>
<td>36</td>
<td>12</td>
<td>0.805</td>
<td>0.001</td>
<td>0.805</td>
<td>0.001</td>
</tr>
<tr>
<td>60</td>
<td>12</td>
<td>0.712</td>
<td>0.092</td>
<td>0.712</td>
<td>0.092</td>
</tr>
<tr>
<td>84</td>
<td>12</td>
<td>0.699</td>
<td>0.033</td>
<td>0.699</td>
<td>0.033</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>0.791</td>
<td>&lt;0.001</td>
<td>0.791</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>60</td>
<td>36</td>
<td>0.819</td>
<td>&lt;0.001</td>
<td>0.819</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>84</td>
<td>36</td>
<td>0.806</td>
<td>&lt;0.001</td>
<td>0.806</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
<td>0.814</td>
<td>&lt;0.001</td>
<td>0.814</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>84</td>
<td>60</td>
<td>0.768</td>
<td>&lt;0.001</td>
<td>0.768</td>
<td>&lt;0.001</td>
</tr>
<tr>
<td>84</td>
<td>84</td>
<td>0.844</td>
<td>&lt;0.001</td>
<td>0.844</td>
<td>&lt;0.001</td>
</tr>
</tbody>
</table>

(a) No Phenotype

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$\delta$</th>
<th>$p$</th>
<th>$\delta_s$</th>
<th>$p_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>0.800</td>
<td>0.078</td>
<td>0.800</td>
<td>0.078</td>
</tr>
<tr>
<td>36</td>
<td>12</td>
<td>0.741</td>
<td>0.01</td>
<td>0.741</td>
<td>0.01</td>
</tr>
<tr>
<td>60</td>
<td>12</td>
<td>0.645</td>
<td>0.078</td>
<td>0.645</td>
<td>0.078</td>
</tr>
<tr>
<td>84</td>
<td>12</td>
<td>0.536</td>
<td>0.106</td>
<td>0.536</td>
<td>0.106</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>0.649</td>
<td>0.003</td>
<td>0.649</td>
<td>0.003</td>
</tr>
<tr>
<td>60</td>
<td>36</td>
<td>0.586</td>
<td>0.001</td>
<td>0.586</td>
<td>0.001</td>
</tr>
<tr>
<td>84</td>
<td>60</td>
<td>0.532</td>
<td>0.001</td>
<td>0.532</td>
<td>0.001</td>
</tr>
<tr>
<td>84</td>
<td>84</td>
<td>0.572</td>
<td>0.001</td>
<td>0.572</td>
<td>0.001</td>
</tr>
</tbody>
</table>

(b) High Variance Phenotype

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$\delta$</th>
<th>$p$</th>
<th>$\delta_s$</th>
<th>$p_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>0.627</td>
<td>0.094</td>
<td>0.627</td>
<td>0.094</td>
</tr>
<tr>
<td>36</td>
<td>12</td>
<td>0.489</td>
<td>0.067</td>
<td>0.489</td>
<td>0.067</td>
</tr>
<tr>
<td>60</td>
<td>12</td>
<td>0.446</td>
<td>0.027</td>
<td>0.446</td>
<td>0.027</td>
</tr>
<tr>
<td>84</td>
<td>12</td>
<td>0.319</td>
<td>0.119</td>
<td>0.319</td>
<td>0.119</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>0.468</td>
<td>0.010</td>
<td>0.468</td>
<td>0.010</td>
</tr>
<tr>
<td>60</td>
<td>36</td>
<td>0.494</td>
<td>0.003</td>
<td>0.494</td>
<td>0.003</td>
</tr>
<tr>
<td>84</td>
<td>36</td>
<td>0.375</td>
<td>0.015</td>
<td>0.375</td>
<td>0.015</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
<td>0.467</td>
<td>0.002</td>
<td>0.467</td>
<td>0.002</td>
</tr>
<tr>
<td>84</td>
<td>60</td>
<td>0.397</td>
<td>0.006</td>
<td>0.397</td>
<td>0.006</td>
</tr>
<tr>
<td>84</td>
<td>84</td>
<td>0.393</td>
<td>0.005</td>
<td>0.393</td>
<td>0.005</td>
</tr>
</tbody>
</table>

(c) Low Variance Phenotype
Figure 3.6: Power plots while varying the variance associated with batch and the phenotype means.

(a) No Phenotype

(b) High Variance Phenotype

(c) Low Variance Phenotype

49
4 Applications

Our test statistic was applied to two case studies. The \( U \) and \( V \) matrices are assumed to be orthogonal \( n \times n \) (or \( b \times b \) for gPCA) and \( p \times p \) matrices, respectively. To adjust for missing values, mean value imputation was performed on the centered data \( X \) prior to PCA. Results were considered significant at a level of \( \alpha = 0.05 \).

4.1 Data

4.1.1 Filtering

For unsupervised learning problems, non-informative features contribute random noise to distance calculations. The resulting effect is that non-informative features mask useful information provided by informative features. Therefore, non-informative features should be assigned a zero weight in the clustering algorithm [23]. The simplest implementation is to exclude identified non-informative features in the clustering analysis. This filtering step is applied to genomic data to remove sources of obscuring variation prior to applying a clustering algorithm. In our simulation studies, we observed higher power when the proportion of features affected by batch increased, therefore, we filtered our data stringently to keep the most variable or informative features. A variance filter was applied to the data to remove noise and reduce the number of features. The standard deviation of each feature was calculated and the 1000 most variable features were retained [7, 12, 19].

The sensitivity of guided principal components analysis (gPCA) to different levels of variance filtering were investigated, allowing the number of features retained by the variance filter to range between 500 and the full GENEMAM data set (657,366 features). A further analysis implementing an ANOVA filter was also investigated. The \texttt{limma} package was used
to fit an ANOVA (lmFit()) model with phenotype represented in the design matrix. The eBayes() function was subsequently used to compute the moderated $F$ statistics and create an indicator of significant features to be used to filter the centered, mean-value imputed data. The methods of Benjamini and Hochberg [3] and Bonferroni were used to adjust for multiple comparisons at significance levels of $\alpha = 0.05$ and 0.01. The main goal of filtering in our analyses is to remove non-informative features and to reduce the time required for the analysis and corresponding permutations. An run time analysis is provided in section 4.2.4.

4.1.2 GENEMAM

The GENetic Epidemiology of MAMmographic Density (GENEMAM) study data included 614 samples from the Minnesota Breast Cancer family study [36]. These samples were genotyped using the Illumina Human 660 bead-chip array. Samples were processed over three time periods on 8 plates. Forty-two samples failed quality-control checks from plates 1-4 due to an Illumina reagent problem and these samples were replated on plate 5, along with 6 other samples. Samples on plates 6-8 were genotyped at a later date. This effectively yielded three batches corresponding to the three different runs. Data for all chromosomes were used. Illumina’s GenomeStudio software was used to obtain the Log$_2$ $R$ ratio (LRR) values. LRR is a measure of relative intensity where $R$ is the sum of the normalized allelic probe intensities produced by SNP assays and the ratio is of observed $R$ divided by the expected value [25].

4.1.3 GENOA

The GENOA data included 1,418 of the non-Hispanic white adults enrolled in the Genetic Epidemiology Network of Arteriopathy (GENOA) study of the Family Blood Pressure Program (FBPP), a study designed to identify germline genetic determinants of hypertension in
multiple ethnic groups. These samples were genotyped on Affymetrix SNP Array 6.0 chips and all samples had contrast QC values greater than 0.4. The PennCNV-Affy Protocol\textsuperscript{1} was followed to obtain the LRR values. The analysis focused on chromosome 22 data using the first 10 plates consisting of 703 samples.

4.2 Results

4.2.1 GENEMAM

The standard use of PCA is to look at the plot of the first principal component of the data \((n \times p\) matrix \(X\) where \(n\) denotes sample and \(p\) denotes probe) versus the second principal component (Figure 4.2a). The GENEMAM data has an obvious batch effect and the PCA plot of the first two principal components shows that this batch effect is due to plate when colored by plate with three batches consisting of plates 1-4, 5, and 6-8. As is common with batch effects, this batch effect is due to the plates being run at different times.

Next, we performed a gPCA with plate as the batch indicator. The gPCA plot of the first two principal components (Figure 4.2b) shows greater separation in the batches, especially of plate 3 from plates 1, 2 and 4, than the unguided principal component plot (Figure 4.2a), but shows the same groupings of plates for these data. After filtering out all but the \(p = 1000\) most variable features, our permutation test confirms that there is a significant batch effect separating the plates \((\delta = 0.5987; p - \text{value} < 0.001)\). Of the variance due to features in these data 87.3\% of the total variation is explained by batch.

Physical sample well location on each plate was looked at as a potential source of poor quality for each plate. Figure 4.2 shows heatmaps of PC\(_1\) and PC\(_2\) for gPCA based on the sample well location for each sample on the 96-well plates. The colors represent the value of

\footnotetext{\textsuperscript{1}http://www.openbioinformatics.org/penncnv/penncnv\_tutorial\_affy\_gw6.html}
Figure 4.1: GENEMAM - (a) Unguided PCA of $X$ of $Y'X$. Samples for each plate are denoted by a different color and/or symbol.

(a) Unguided PCA

the principal component for that sample and the scale for the colors is standardized across all plates. These plots show that PC$_1$ separates plates 6 through 8, from 5, and from 1 through 4. PC$_2$ separates the first half of plate 3 from all the rest. This analysis identifies potential quality issues with plate 3. White space on the plates in the heatmaps represents missing
This case study is an example with an obvious batch effect and thus did not require specialized methods to detect since batch was the largest source of variability.
Figure 4.2: GENEMAM - Standardized Heatmaps showing the (a) $PC_1$ values at each sample well location. White spaces indicate missing samples for the plate. Plates 5 and 8 were incomplete plates.

4.2.2 GENOA

In this case study, batch is not so easily detected using unguided PCA. Unguided PCA was performed and Figure 4.4a shows the PCA plot of the first two principal components. Figure
Figure 4.2: GENEMAM - Standardized Heatmaps showing the (b) PC$_2$ values at each sample well location. White spaces indicate missing samples for the plate. Plates 5 and 8 were incomplete plates.

4.4a shows that plates 7 and 8 might be slightly separated from the rest of the plates. A gPCA with batch indicated by plate (Figure 4.4b) shows that plates 7 and 8 along with plate 4 separate slightly from the other plates. It is not obvious from the unguided PCA on
\( X \) that plate 4 is separate from the rest of the plates. However, gPCA shows a separation between 4 and the rest of the plates. After filtering out all but the \( p = 1000 \) most variable features, our permutation test shows that there is a significant batch effect separating the plates \((\delta = 0.9219; p\text{-}value < 0.001)\). Of the variance due to SNPs in this data 71\% of the total variation is explained by batch. gPCA identifies a batch (plate 4) that does not otherwise stand out in an unguided principal component plot.

4.2.3 Sensitivity of gPCA Results to Filtering

We also performed a sensitivity analysis allowing the number of features retained by the variance filter to range between 10 and the full GENEMAM data set. We also implemented an ANOVA filter where feature-level linear models were fit where the batch indicators were predictors and the overall \( F \)-test were used. Features were considered significant if their Benjamini and Hochberg [3] adjusted \( p \)-value was < 0.05.

The GENEMAM and GENOA case study data were filtered using a variance filter to retain the 1000 most variable features. The sensitivity of the results of gPCA to this filtering was investigated using the both data sets and simulated data. Table 4.1 shows the resulting \( p \)-values from retaining between 10 and all features from the full (a) GENEMAM data set or (b) GENOA data set. For the GENEMAM data, as long as 7.6\% (500 features) or more features are retained, significant batch effects are found. Since filtering to retain 500 features takes approximately 1 minute to run, there is no need to retain fewer features. For the GENOA data, as long as 21.6\% (100 features) or more features are retained, significant batch effects are found, which takes approximately 8 seconds to run. We found that gPCA is not sensitive to filtering for the application datasets and thus filtering can be used as a method to reduce the analysis time required provided it is judiciously applied (Table 4.1).
Table 4.1: Variance Filtering Sensitivity Results: $\delta$, corresponding $p$-values resulting from retaining between 10 and all features from the full data set. The last column gives the system time in minutes required to run gPCA as discussed in Section 4.2.4.

<table>
<thead>
<tr>
<th>Features</th>
<th>% Features</th>
<th>$\delta$</th>
<th>$p$-value</th>
<th>System Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.002</td>
<td>0.6878</td>
<td>0.511</td>
<td>0.812</td>
</tr>
<tr>
<td>20</td>
<td>0.003</td>
<td>0.5617</td>
<td>0.706</td>
<td>0.779</td>
</tr>
<tr>
<td>50</td>
<td>0.008</td>
<td>0.6129</td>
<td>0.119</td>
<td>0.841</td>
</tr>
<tr>
<td>100</td>
<td>0.015</td>
<td>0.4603</td>
<td>0.264</td>
<td>0.866</td>
</tr>
<tr>
<td>200</td>
<td>0.030</td>
<td>0.4194</td>
<td>0.268</td>
<td>0.892</td>
</tr>
<tr>
<td>500</td>
<td>0.076</td>
<td>0.5428</td>
<td>0.012</td>
<td>0.965</td>
</tr>
<tr>
<td>1000</td>
<td>0.152</td>
<td>0.5987</td>
<td>$&lt; 0.001$</td>
<td>1.144</td>
</tr>
<tr>
<td>2000</td>
<td>0.304</td>
<td>0.6914</td>
<td>$&lt; 0.001$</td>
<td>1.453</td>
</tr>
<tr>
<td>5000</td>
<td>0.761</td>
<td>0.7244</td>
<td>$&lt; 0.001$</td>
<td>2.479</td>
</tr>
<tr>
<td>10000</td>
<td>1.521</td>
<td>0.8344</td>
<td>$&lt; 0.001$</td>
<td>3.895</td>
</tr>
<tr>
<td>20000</td>
<td>3.042</td>
<td>0.9814</td>
<td>$&lt; 0.001$</td>
<td>7.620</td>
</tr>
<tr>
<td>50000</td>
<td>7.606</td>
<td>0.9807</td>
<td>$&lt; 0.001$</td>
<td>15.348</td>
</tr>
<tr>
<td>100000</td>
<td>15.212</td>
<td>0.9819</td>
<td>$&lt; 0.001$</td>
<td>33.395</td>
</tr>
<tr>
<td>200000</td>
<td>30.424</td>
<td>0.9835</td>
<td>$&lt; 0.001$</td>
<td>60.809</td>
</tr>
<tr>
<td>500000</td>
<td>76.061</td>
<td>0.9839</td>
<td>$&lt; 0.001$</td>
<td>162.075</td>
</tr>
<tr>
<td>657366</td>
<td>100.000</td>
<td>0.9839</td>
<td>$&lt; 0.001$</td>
<td>206.657</td>
</tr>
</tbody>
</table>

(a) GENEMAM

<table>
<thead>
<tr>
<th>Features</th>
<th>% Features</th>
<th>$\delta$</th>
<th>$p$-value</th>
<th>System Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.043</td>
<td>0.8664</td>
<td>0.087</td>
<td>0.118</td>
</tr>
<tr>
<td>20</td>
<td>0.087</td>
<td>0.8117</td>
<td>0.063</td>
<td>0.118</td>
</tr>
<tr>
<td>50</td>
<td>0.216</td>
<td>0.7693</td>
<td>0.025</td>
<td>0.129</td>
</tr>
<tr>
<td>100</td>
<td>0.433</td>
<td>0.7421</td>
<td>0.008</td>
<td>0.146</td>
</tr>
<tr>
<td>200</td>
<td>0.865</td>
<td>0.8315</td>
<td>$&lt; 0.001$</td>
<td>0.183</td>
</tr>
<tr>
<td>500</td>
<td>2.163</td>
<td>0.9220</td>
<td>$&lt; 0.001$</td>
<td>0.302</td>
</tr>
<tr>
<td>1000</td>
<td>4.326</td>
<td>0.9219</td>
<td>$&lt; 0.001$</td>
<td>0.520</td>
</tr>
<tr>
<td>2000</td>
<td>8.652</td>
<td>0.9006</td>
<td>$&lt; 0.001$</td>
<td>0.977</td>
</tr>
<tr>
<td>5000</td>
<td>21.631</td>
<td>0.8811</td>
<td>$&lt; 0.001$</td>
<td>2.394</td>
</tr>
<tr>
<td>10000</td>
<td>43.262</td>
<td>0.8620</td>
<td>0.006</td>
<td>4.052</td>
</tr>
<tr>
<td>20000</td>
<td>86.524</td>
<td>0.8338</td>
<td>0.012</td>
<td>8.051</td>
</tr>
<tr>
<td>23115</td>
<td>100.000</td>
<td>0.8282</td>
<td>0.013</td>
<td>9.388</td>
</tr>
</tbody>
</table>

(b) GENOA
Figure 4.3: GENOA - (a) Unguided PCA of $X$. Samples for each plate are denoted by a different color and/or symbol.

ANOVA Filtering An analysis of variance (ANOVA) filter was applied to the GENEMAM data to assess it as an alternative to variance filtering. Table 4.2 shows the number of features retained from each adjustment method. In all cases the number of features is very large owing to the large batch effect present in this dataset and the data being only globally normalized.
Figure 4.3: GENOA - (b) Guided PCA of $Y'X$. Samples for each plate are denoted by a different color and/or symbol.

As shown in section 4.2.3, gPCA is not sensitive to filtering, so filtering can be used to reduce the data dimension and facilitate implementing gPCA by reducing the analysis time without worry.
Table 4.2: Number of features retained using an ANOVA filtering method with different multiple comparison adjustment methods and stringencies.

<table>
<thead>
<tr>
<th>Adj. Method</th>
<th>$\alpha$</th>
<th>Feat. Retained</th>
</tr>
</thead>
<tbody>
<tr>
<td>BH</td>
<td>0.05</td>
<td>636141</td>
</tr>
<tr>
<td>BH</td>
<td>0.01</td>
<td>624797</td>
</tr>
<tr>
<td>Bonferroni</td>
<td>0.05</td>
<td>546012</td>
</tr>
<tr>
<td>Bonferroni</td>
<td>0.01</td>
<td>535708</td>
</tr>
</tbody>
</table>

4.2.4 gPCA Run Time Analysis

An analysis of the time it takes to run gPCA on varying sizes of data was performed. Table 4.1(a) gives the time it takes to run gPCA on the GENEMAM data with between 10 and the full set of features. There were $n = 614$ samples and the data was centered and mean-value imputed prior to performing gPCA. Table 4.1(b) gives the time it takes to run gPCA on the GENOA data with between $p = 10$ and the full set of features. There were $n = 703$ samples and the data was not centered prior to performing gPCA. There were no missing values so mean-value imputation was not necessary.

As can be seen in these analyses, batch effects can be a prominent source of variation in high-throughput genomic data. Our new statistic $\delta$ uses gPCA to successfully test for batch effect. In Chapter 5, we demonstrate that our test statistic can be used to evaluate the performance of batch correction methods.

4.2.5 Analysis of Batch and Phenotype Confounding

An analysis of the effects of confounding of batch and phenotype was performed to assess how the gPCA $\delta$ statistic performs and to show the effects of confounding on batch correction. The GENEMAM data was used with run time considered to be batch and plate considered to be phenotype after filtering to retain the 1000 most variable features. Run time and plate are highly confounded (Pearson’s $\rho = 0.9$) since plates 1-4, plate 5, and plates 6-8 were run
in three separate batches at different times.

After fitting a linear model using the `lmFit()` function with plate as phenotype as the predictor, the number of significant features in the GENEMAM data was assessed using the `eBayes()` function in the `limma` package both prior to batch correction and after batch correction using the batch-mean centering (BMC) method of Sims et al. [38]. For batch correction, BMC was implemented using the `pamr.batchadjust()` function in the `pamr` package.

Table 4.3 provides a contingency table of the results. Prior to batch correction 191 of the 1000 features were found to be significant in terms of the phenotype and a significant batch effect was found ($\delta = 0.5828; p$-value < 0.001), but after batch correction using run time as batch, only 7 features were found to be significant and no significant batch effect was found ($\delta = 0.044; p$-value = 1). This indicates that if a phenotype of interest is confounded with batch, any batch correction procedure would remove that variation from the data resulting in finding no significant features associated with that phenotype.

Table 4.3: Results of Confounding Batch and Phenotype: Number of significant features prior to and post-batch correction using BMC when batch and phenotype are confounded. Rows give the results of the test prior to batch correction and the columns give results of the test post-batch correction. The “Reject” column and row indicate the number of features that are significantly predicted by phenotype.

<table>
<thead>
<tr>
<th></th>
<th>Fail to reject</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail to reject</td>
<td>809</td>
<td>0</td>
</tr>
<tr>
<td>Reject</td>
<td>184</td>
<td>7</td>
</tr>
</tbody>
</table>
5 Comparison of Batch Effect Adjustment Methods

5.1 Introduction

Batch effects are commonly observed systematic non-biological variation between groups of samples due to experimental artifacts, such as processing date, lab, or technician. Combining samples from multiple batches can cause the true biological variation in a high-throughput experiment to be obscured by variation due to batch. Global normalization methods such as ANOVA [10], loess-based methods [12], and quantile [4] correct for experimental artifacts that effect all probes similarly. However, post-global normalization probe-specific batch effects are commonly present, as often detected by principal components analysis (PCA). Many methods have been developed to correct high-throughput data for batch effects including normalization methods [39], frozen robust multiarray average (fRMA) for gene expression microarray data [33], orthogonal projections to latent structures (OPLS) for multi-channel microarray data [5], corrected robust linear models with maximum likelihood classification version 2 (CRLMMv2) for SNP-based microarray platforms [6], surrogate variable analysis (SVA) for gene expression microarray data [27], batch mean-centering (BMC) for microarray data [31, 38], distance weighted discrimination (DWD) for microarray data [2, 17, 18, 32], and empirical Bayes (EB) for gene expression and CNV data [22]. Many of these methods have been previously reviewed by Chen et al. [8], Lazar et al. [26], and Luo et al. [31]. These methods were introduced in Chapter 1.

Previous studies have found that BMC, DWD, and EB generally outperform all other methods [8, 26]. Both BMC and EB produce batch corrected data sets. After further data manipulation according to the methods of Benito et al. [2] and Huang et al. [18], the results
of DWD can also be converted into a batch corrected matrix. In this chapter we will apply these three methods to two case studies that have known significant batch effects and to three simulated data sets, and assess their usefulness at batch effect removal using our test statistic $\delta$ that employs guided principal components analysis (gPCA) [34].

5.2 Statistical Methods

5.2.1 Guided Principal Components Analysis

Introduced in Chapter 2, guided principal components analysis (gPCA) is an extension of principal components analysis (PCA) that replaces the data $X$ matrix in the singular value decomposition (SVD) of PCA with $Y'X$ such that

$$Y'X = UDV'$$

where $Y$ is an $n \times b$ indicator matrix where $n$ denotes sample and $b$ denotes batch. Each of the $k = 1, \ldots, b$ batches is comprised of $n_k$ observations such that $\sum_k n_k = n$. The indicator matrix consists of $b$ blocks with $n_k$ rows for $k = 1, \ldots, b$, and $k$ columns where, for each block,

$$Y_k = \begin{cases} 1 & \text{if } k = b \\ 0 & \text{otherwise} \end{cases}.$$ 

Performing SVD on $Y'X$ results in a $b \times b$ batch loadings matrix $U$ and a $p \times p$ probe loadings matrix $V$. Large singular values (the diagonal elements of the $q \times q$ matrix $D$ where $q = \min(n, p)$) imply that the batch is important for the corresponding principal component. gPCA guides the SVD to look for batch effects in the data based on the batch indicator matrix $Y$, which can be defined to indicate any type of potential batch effect.

In Chapter 2, we proposed a test statistic $\delta$ that quantifies the proportion of variance
due to batch effects in experimental genomic data. The proportion of total variance due to
the first principal component is taken to be the ratio of the variance of the first principal
component from gPCA to the variance of the first principal component from unguided PCA

\[ \delta = \frac{\text{var}(XV_g)}{\text{var}(XV_u)} \]

where \( g \) indicates gPCA and \( u \) indicates unguided PCA. \( V \) is the matrix of probe loadings
resulting from gPCA or PCA, respectively. Large values of \( \delta \) (values near 1) imply that the
batch effect is large.

To determine whether \( \delta \) is significantly larger than would be expected by chance, a \( p \)-
value is estimated using a permutation distribution created by permuting the batch vector
\( M = 1000 \) times so that \( \delta_{pm} \) is computed for \( m = 1, \ldots, M \) where \( p \) indicates the permutation.
Here \( \delta_{pm} \) is the proportion of the total variance due to the first principal component from the
\( m^{th} \) permutation from gPCA to the total variance due to the first principal component from
the \( m^{th} \) permutation from unguided PCA. A one-sided \( p \)-value (testing \( H_0 : \delta_{pm} = \delta \) versus
\( H_1 : \delta_{pm} > \delta \)) is estimated as the proportion of times the observed \( \delta \) was in the extreme tail
of the permutation distribution

\[ p\text{-value} = \frac{\sum_{m=1}^{M} (\delta_{pm} > \delta)}{M}. \]

For more details on gPCA see Chapter 2.

5.2.2 Batch Mean-Centering

Sims et al. [38] introduced batch mean-centering (BMC) as a means of adjusting high-
throughput genomic data for batch effects. BMC simply centers the data within a batch, so
that the batch means are all zero. This is also referred to as one-way analysis of variance adjustment by Luo et al. [31]. Mathematically, the batch means across each feature $j$ and within each batch $k$ are calculated as

$$
\bar{x}_{jk} = \frac{1}{n_k} \sum_{i=1}^{n_k} x_{ijk}.
$$

The data $x_{ijk}$ are then adjusted by

$$
x_{ijk}^* = x_{ijk} - \bar{x}_{jk}.
$$

Sims et al. [38] found that BMC successfully reduced the amount of between batch (or dataset) variation, while maintaining the within batch variation, allowing multiple batches (or datasets) to be further analyzed together, thus increasing the statistical power of future analyses.

### 5.2.3 Distance Weighted Discrimination

As described in Chapter 1 Section 1.3.6, distance weighted discrimination (DWD) is a classification method developed by Marron and Todd [32] that overcomes the data-piling problems in high dimension low sample size (HDLSS) situations of other methods such as support vector machines (SVM) [2, 32]. In HDLSS data scenarios, DWD improves generalizability as well. The binary classification with DWD is well documented [2, 32]; however, the multiclass case is not as developed. Huang et al. [17, 18] have developed an extension of DWD to multicategory classification called MDWD and provide an R package DWD that includes multiclass DWD functionality. DWD can be used to combine multiple datasets while adjusting for dataset or to adjust for variation due to batch effects. Since our case study datasets have

66
more than two batches, the multiclass methods are necessary.

**Binary DWD** Marron and Todd [32] introduce DWD, focusing on two class linear discrimination, meaning that “the discrimination rule is a simple linear function of the new data vector.” They introduce a direction vector \( w \) and threshold \( \beta \) such that the new data vector \( x \) is assigned to the positive class (+1) when \( x'w + \beta \geq 0 \), where the two classes have labels +1 and −1. Marron and Todd [32] introduce a new optimization method that optimizes the sum of the inverse distances from the data to the separating hyperplane which allows the distances, \( r_i \), to influence the direction vector \( w \).

Marron and Todd [32] let the training data consist of \( n \) vectors \( x_i \) of length \( p \) with corresponding class indicators \( y_i \) where class is indicated by +1 or −1. Then \( X \) is the \( p \times n \) matrix with columns \( x_i \) and \( y \) is a length \( n \) vector indicating the sample classes. The number of samples in each class can be written \( n_+ = \sum_{i=1}^{n} 1\{y_i = +1\} \) and \( n_- = \sum_{i=1}^{n} 1\{y_i = -1\} \) so that \( n = n_+ + n_- \). Marron and Todd [32] let \( Y \) be a \( n \times n \) diagonal matrix with \( y \) on the diagonal. Then, they choose the direction vector (or normal vector) to be \( w \in \mathbb{R}^p \) as the hyperplane and the position to be \( \beta \in \mathbb{R} \). The residual of the \( i \)th data point is then

\[
\bar{r}_i = y_i(x_i'w + \beta)
\]

or, in matrix notation,

\[
\bar{r} = Y(X'w + \beta e) = YX'w + \beta y
\]

where \( e \) is a length \( n \) vector of ones. Ideally, \( w \) and \( \beta \) would be chosen such that the residuals are all positive and relatively large. The vector “\( w \) is scaled to have unit norm so that the residuals measure the signed distances of the points from the hyperplane.” Since the positive
and negative data might not be able to be separated linearly, an error vector $\xi \in \mathbb{R}^n_+$ is added (and penalized) and the perturbed residuals are

$$r = YX'w + \beta y + \xi$$

Marron and Todd [32] discuss the optimization problem for the DWD approach in depth. In short, they minimize the sum of the reciprocals of the residuals, perturbed by a penalized vector $\xi$, such that

$$\min_{r, w, \beta, \xi} \sum_i (1/r_i) + C e' \xi, \quad r = YX'w + \beta y + \xi, \quad (1/2)w'w = 1/2, \quad r \geq 0, \quad \xi \geq 0$$

where $C > 0$ is a penalty parameter. They further apply a second-order cone programing (SCOP) problem, an interior-point method, for optimization. Further details on the optimization methods and choice of the tuning parameter $C$ can be found in Marron and Todd [32].

Benito et al. [2] provide methodology to produce batch corrected data matrices using DWD. Their methodology for batch adjustment for binary classes is a) find the DWD direction vector $w$; b) project subpopulations in the DWD direction, $(v_+ = x_+ w$ and $v_- = x_- w)$; c) compute projected subpopulation means $(\mu_+ = \sum v_+ / n_+$ and $\mu_- = \sum v_- / n_-)$; and d) shift each subpopulation in the DWD direction by an appropriate amount (found by subtracting the DWD direction vector multiplied by each projected mean for each gene; $x^*_+ = x_+ - \mu_+ we$ and $x^*_- = x_- - \mu_- we$ where $e$ is a length $n$ vector of ones). This produces a batch corrected data matrix $x^*$ which can be used in further analyses.
Multiclass DWD Huang et al. [18] discuss extending binary DWD to the multiclass case in their forthcoming paper. They discuss multiple strategies that account for more than two classes by solving a series of binary problems using One-Versus-One (OVO) or One-Versus-The-Rest (OVR) approaches of Duda et al. [11] and Hastie et al. [15], respectively, and introduce a new method that accounts for multiple classes globally. Huang et al. [18]’s multiclass DWD (MDWD) method address the $b$ class problem which simultaneously produces $b$ direction vectors. These direction vectors provide the basis of their batch adjustment method. “The $b$ normal direction vectors determine a subspace which contains each class mean. [They] move each class in such a way that the class means move to a common point in this subspace.” The batch adjustment procedure of Huang et al. [18] is

a) find the $p \times n$ matrix of MDWD direction vectors $w$ which generates a subspace $V$;

b) project the subpopulations (e.g. respective batch subsets) onto that subspace ($P_{V_k} = X_k w$ where $P_{V_k}$ is $n_k \times b$ for each $k = 1, \ldots, b$);

c) compute the coordinates of the subpopulation projected means ($\mu_{P_{V_k}} = \frac{1}{n_k} \sum_{i=1}^{n_k} P_{V_{ki}}$), essentially, the column means of the projection matrix $P_{V_k}$ for each batch $k$; and

d) shift each subpopulation such that its projected mean is moved in the subspace to a fixed point which is common to all subpopulations ($X_k^* = X_k - (w \mu_{P_{V_k}} e_{n_k})'$ where $e_{n_k}$ is a $1 \times n_k$ matrix of ones and $w \mu_{P_{V_k}}$ is an $p \times 1$ matrix of the direction matrix multiplied by the projected means for batch $k$).

Since the MDWD direction vectors maximize the separation between the batches and ignore the variation in the data, MDWD preserves the variation that is not due to batch effects. Further details on the optimization process of Huang et al. [18] can be found in their paper.
5.2.4 Empirical Bayes

Empirical Bayes (EB) methods have long been applied to microarray data analysis due to “their ability to robustly handle high-dimensional data when sample sizes are small” [22]. Johnson et al. [22] extend the EB methods to adjust for batch effects in microarray data and provide both parametric and non-parametric shrinkage adjustments. Parametric Bayes assumes data follows a prior probability distribution, and that the parameters of that distribution themselves follow prior distributions. Non-parametric Bayes assumes that the data follows a prior distribution, but the parameters of the distribution are estimated using the posterior distribution. There are three basic steps to the EB framework, step 2 of which varies between parametric (Step 2a) and non-parametric (Step 2b) methods.

The EB frameworks assume the data have been globally normalized and thus normalized expression values are available for all features and samples. Let the data contain $i = 1, \ldots, n$ samples and $k = 1, \ldots, b$ batches where each batch includes $n_k$ samples and $j = 1, \ldots, p$ features. We assume the model

$$X_{ijk} = \alpha_j + Y\beta_j + \gamma_{jk} + \delta_{jk}\epsilon_{ijk}$$

where $X_{ijk}$ is the normalized expression data, $\alpha_j$ is the overall expression for feature $j$, $Y$ is a design matrix of sample conditions (for example, batch), $\beta_j$ is the vector of regression coefficients corresponding to $Y$ for feature $j$, $\gamma_{jk}$ are the additive batch effects for batch $k$ for feature $j$, and $\delta_{jk}$ are the multiplicative batch effects for batch $k$ for feature $j$. The errors, $\epsilon_{ijk}$, are assumed to be normally distributed with mean zero and variance $\sigma_j^2$. 

70
Step 1: Standardize the data

The first step of the method proposed by Johnson et al. [22] is to account for bias in the EB estimates due to expression magnitude differences across features which could cause $\alpha_j$, $\beta_j$, $\gamma_j$, and $\sigma_j^2$ to vary across features. To avoid the potential for bias from this source, the data are standardized gene-wise to have similar mean and variance as

$$Z_{ijk} = \frac{X_{ijk} - \hat{\alpha}_j - Y\hat{\beta}_j}{\hat{\sigma}_j}$$

where the model parameters $\alpha_j$, $\beta_j$, and $\gamma_j$ have been estimated as $\hat{\alpha}_j$, $\hat{\beta}_j$, and $\hat{\gamma}_j$ for $k = 1, \ldots, b$ and $j = 1, \ldots, p$. Johnson et al. [22] employ gene-wise ordinary least squares to estimate the parameters and to make sure the parameters are identifiable, they constrain $\sum_i n_i \hat{\gamma}_{jk} = 0$ for all $j = 1, \ldots, p$. The variance can the be estimated as $\hat{\sigma}_j^2 = \frac{1}{N} \sum_{ik} (X_{ijk} - \hat{\alpha}_j - Y\hat{\beta}_j - \hat{\gamma}_{jk})^2$ where $N$ is the total number of samples ($N = \sum_k n_k$).

Step 2a: EB batch effect parameter estimates using parametric empirical priors

The standardized data are assumed to satisfy $Z_{ijk} \sim N(\gamma_{jk}, \delta_{jk}^2)$. The parametric forms of the prior distributions of the batch effect parameters are assumed to be

$$\gamma_{jk} \sim N(X_k, \tau_k^2) \quad \text{and} \quad \delta_{jk}^2 \sim \text{Inverse Gamma}(\lambda_k, \theta_k)$$

where the hyperparameters $\gamma_k$, $\tau_k^2$, $\lambda_k$, and $\theta_k$ are estimated empirically using the method of moments from the standardized data. Johnson et al. [22] chose these prior distributions due to their conjugacy with the Normal assumption of the standardized data. Based on the above distributional assumptions, the EB estimates for the batch effects parameters $\gamma_{jk}$ and
\( \delta_{jk}^2 \) are given by the conditional posterior means

\[
\gamma_{jk}^* = \frac{n_k \bar{\gamma}_{jk}^2 + \delta_{jk}^2}{n_k \bar{\gamma}_{jk}^2 + \delta_{jk}^2} \quad \text{and} \quad \delta_{jk}^2 = \frac{\tilde{\delta}_k + \frac{1}{2} \sum_i (Z_{ijk} - \gamma_{jk}^*)^2}{\frac{n_k}{2} + \lambda_k - 1},
\]

respectively.

**Step 2b: EB batch effect parameter estimates using non-parametric empirical priors** The standardized data are assumed to satisfy \( Z_{ijk} \sim N(\gamma_{jk}, \delta_{jk}^2) \) as above. We further assume

\[
\hat{\gamma}_{jk} = \frac{1}{n_k} \sum_i Z_{ijk} \quad \text{and} \quad \hat{\delta}_{jk}^2 = \frac{1}{n_k - 1} \sum_i (Z_{ijk} - \hat{\gamma}_{jk})^2.
\]

The batch effect parameters \( \gamma_{jk} \) and \( \delta_{jk}^2 \) are then estimated using estimates of the posterior expectations of the batch effect parameters, \( E[\gamma_{jk}] \) and \( E[\delta_{jk}^2] \). We let \( Z_{jk} \) be a vector containing \( Z_{ijk} \) for \( i = 1, \ldots, n_k \). Then the posterior expectation of \( \gamma_{jk} \) is

\[
E[\gamma_{jk}] = \int \gamma_{jk} \pi(Z_{jk}, \gamma_{jk}, \delta_{jk}^2) d(\gamma_{jk}, \delta_{jk}^2)
\]

given the posterior distribution \( \pi(Z_{jk}, \gamma_{jk}, \delta_{jk}^2) \) of the data \( Z_{jk} \) and the batch effect parameters \( \gamma_{jk} \) and \( \delta_{jk}^2 \). Let the unspecified density function for the prior for the parameters \( \gamma_{jk} \) and \( \delta_{jk}^2 \) be \( \pi(\gamma_{jk}, \delta_{jk}^2) \) and let the likelihood \( L(Z_{jk} \mid \gamma_{jk}, \delta_{jk}^2) = \prod_i \varphi(Z_{ijk}, \gamma_{jk}, \delta_{jk}^2) \) where \( \varphi(Z_{ijk}, \gamma_{jk}, \delta_{jk}^2) \) is the probability density function (pdf) of a random variable distributed \( N(\gamma_{jk}, \delta_{jk}^2) \) and evaluated at \( Z_{ijk} \). Equation 5.1 above can then be written

\[
E[\gamma_{jk}] = \frac{1}{C(Z_{jk})} \int \gamma_{jk} L(Z_{jk} \mid \gamma_{jk}, \delta_{jk}^2) \pi(\gamma_{jk}, \delta_{jk}^2) d(\gamma_{jk}, \delta_{jk}^2)
\]
where \( C(Z_{jk}) = \int L(Z_{jk} | \gamma_{jk}, \delta^2_{jk}) \pi(\gamma_{jk}, \delta^2_{jk}) d(\gamma_{jk}, \delta^2_{jk}) \). Johnson et al. [22] then estimated both \( C(Z_{jk}) \) and the integral in 5.2 using Monte Carlo integration using the empirically estimated \((\gamma_{jk}, \delta^2_{jk})\) pairs. These pairs are considered random selections from \( \pi(\gamma_{jk}, \delta^2_{jk}) \).

Finally, if we let \( w_{jk''} = L(Z_{jk} | \hat{\gamma}_{jk''}, \hat{\delta}^2_{jk''}) \) for \( j'' = 1, \ldots, p \), then we can estimate \( C(Z_{jk}) \) as \( \hat{C}(Z_{jk}) = \frac{1}{n} \sum_{j''} w_{jk''} \)

\[ \hat{\gamma}_{jk} = \hat{E}[\gamma_{jk}] = \frac{\sum_{j''} w_{jk''} \hat{\gamma}_{jk''}}{n \hat{C}(Z_{jk})} \]

The same method is used to find the posterior expectation of \( \delta^2_{jk} \). The non-parametric EB batch adjustments are then given by

\[ \hat{\gamma}^*_{jk} = \frac{\sum_{j''} w_{jk''} \hat{\gamma}_{jk''}}{\sum_{j''} w_{jk''}} \quad \text{and} \quad \hat{\delta}^2_{jk} = \frac{\sum_{j''} w_{jk''} \hat{\delta}^2_{jk''}}{\sum_{j''} w_{jk''}} \]

**Step 3: Adjust the data for batch effects** The data can now be adjusted using the EB estimated batch effect parameters as

\[ \hat{\gamma}^*_{ijk} = \frac{\hat{\sigma}_j}{\hat{\delta}^2_{jk}} (Z_{ijk} - \hat{\gamma}^*_{jk}) + \hat{\alpha}_j + Y \hat{\beta}_j \]

Our test statistic \( \delta \) can then be applied to the EB batch corrected data \( \hat{\gamma}^*_{ijk} \) to test whether EB batch correction successfully corrected the data for batch effects or not.

**5.2.5 Evaluation of Batch Effect Correction Methods**

To determine if a correction method successfully removed batch effects from the data, our test statistic \( \delta \) can be applied where \( X \) in \( Y'X \) is replaced by the batch corrected matrix of data. Of the many batch effect correction methods available, few provide a matrix of batch
effect corrected data. Although global normalization methods such as fRMA, quantile, and loess do yield normalized data, they do not correct for probe-specific batch effects. Therefore, herein, we assess BMC, DWD, and EB and test if they work by using our gPCA test statistic.

5.3 Application Data

5.3.1 Simulation Study

Most often investigators are interested in modeling their data in the presence of a known phenotype. Therefore, we simulated data to represent copy number data under three scenarios: (1) feature data (here, feature denotes probe) with no phenotypic variable; (2) feature data with a high variance phenotypic variable; and (3) feature data with a low variance phenotypic variable. As described in Chapter 2, the feature data were generated independently from a normal distribution with 1000 features and 90 observations. Data with two batches and two phenotypes were simulated. The proportion of features affected by batch was 0.010 for the no phenotype scenario, 0.03 for the high variance phenotype scenario, and 0.05 for the low variance phenotype scenario. Batch means \( \mu_{b_1} = 0 \) and \( \mu_{b_2} = 1 \) and batch variance \( \sigma_b^2 = 0.5 \) were used to simulate the data. For the scenarios with phenotypic effects, the proportion of features affected by phenotype was \( p_{\text{prop}} = 0.1 \) and phenotypic means were \( \mu_{p_1} = 0 \) and \( \mu_{p_2} = 1 \). The phenotypic variance was \( \sigma_p^2 = 2 \) for the high variance phenotype scenario and \( \sigma_p^2 = 0.2 \) for the low variance phenotype scenario. In all scenarios, batch effect was simulated independently of phenotype effect. The gPCA test statistic was applied to these three simulated data sets before and after batch correction using the four batch correction methods discussed above to evaluate the presence of a batch effect.

An additional simulation was performed that simulated data with a phenotypic effect so that batch effect and phenotype effect were not independent. These data were simulated
such that each feature had either no effect, a batch effect, a phenotypic effect, or a batch and phenotypic effect. We simulated the data so that 50 features had a batch effect, 50 features had a phenotypic effect, and 100 features had both a batch and phenotypic effect, leaving 800 features with no effect. For feature $j$, we let

$$f_j = \beta_p p_j \text{pheno} + \beta_b b_j \text{batch} + e$$

where $p$ and $b$ are length $p$ vectors indicating whether each feature had a phenotypic or batch effect, respectively, $\text{pheno}$ and $\text{batch}$ are length $n$ vectors giving the phenotype and batch effect for each sample, and $e \sim N(0, \sigma_b)$ is a random error term. The $\beta_p$ and $\beta_b$ parameters determine the magnitude of the phenotypic and batch effects, respectively. If feature $j$ has both a phenotype effect and a batch effect then $p = b = 1$, if $j$ has only a phenotype effect then $p = 1$ and $b = 0$, if $j$ has only a batch effect then $p = 0$ and $b = 1$, and finally if $j$ has neither effect then $p = b = 0$. The $f_j$ feature vectors for $j = 1, \ldots, p$ form our $n \times p$ feature data $X$. For our simulation, $\beta_p = 0.5$ and $\beta_b = 2$, so the batch effect on the features was far greater than the phenotypic effect. An assessment of the sensitivity to batch correction was applied to these simulated data as well as the other three simulated data sets.

### 5.3.2 Case Studies

The four batch correction methods were also applied to two case studies. To adjust for missing values, mean value imputation was performed on the centered data $X$ prior to PCA.

**Filtering** For unsupervised learning problems, non-informative features contribute random noise to distance calculations. The resulting effect is that non-informative features mask useful information provided by informative features. Therefore, non-informative features should be assigned a zero weight in the clustering algorithm [23]. The simplest implementation for
assigning a non-zero weight in a cluster analysis is to exclude identified non-informative features. This filtering step is applied to genomic data to remove sources of obscuring variation prior to applying a clustering algorithm. In our simulation studies, we observed higher power when the proportion of features affected by batch increased, therefore, we filtered our case study data stringently to keep the most variable or informative features. A variance filter was applied to the data to remove noise and reduce the number of features. The standard deviation of each feature was calculated and the 1000 most variable features were retained [7, 12, 19].

**GENEMAM** The GENetic Epidemiology of MAMmographic Density (GENEMAM) study data included 614 samples from the Minnesota Breast Cancer family study [36]. These samples were genotyped using the Illumina Human 660 bead-chip array. Samples were processed over three time periods on 8 plates. Forty-two samples failed quality-control checks from plates 1-4 due to an Illumina reagent problem and these samples were replated on plate 5, along with 6 other samples. Samples on plates 6-8 were genotyped at a later date. This effectively yielded three batches corresponding to the three different runs. Data for all chromosomes were used. Illumina’s GenomeStudio software was used to obtain the Log$_2$ $R$ ratio (LRR) values. LRR is a measure of relative intensity where $R$ is the sum of the normalized allelic probe intensities produced by SNP assays and the ratio is of observed $R$ divided by the expected value [25].

**GENOA** The GENOA data included 1,418 of the non-Hispanic white adults enrolled in the Genetic Epidemiology Network of Arteriopathy (GENOA) study of the Family Blood Pressure Program (FBPP), a study designed to identify germline genetic determinants of hypertension in multiple ethnic groups. These samples were genotyped on Affymetrix SNP
Array 6.0 chips and all samples had contrast QC values greater than 0.4. The PennCNV-Affy Protocol\(^1\) was followed to obtain the LRR values. The analysis focused on chromosome 22 data using the first 10 plates consisting of 703 samples.

### 5.4 Results

Guided principal components analysis (gPCA) was performed on simulated data under three scenarios, no phenotypic effect, dichotomous phenotypic effect, and continuous phenotypic effect, and on the GENEMAM and GENOA case study data. The results of gPCA on the raw data are shown in Table 5.1(a) and all data have a significant batch effect prior to batch effect correction. Four batch correction methods, batch mean-centering (BMC), multiclass distance weighted discrimination (mDWD), non-parametric empirical Bayes (EBn), and parametric empirical Bayes (EBp), were applied to the raw data. gPCA was again performed on the batch corrected data and the results are shown in Table 5.1(b-e). The ComBat software that employs empirical Bayes requires a phenotypic variable which we did not have for the GENOA case study data or, since it was simulated without one, for the no phenotype simulation data. For all data sets, BMC and the two EB methods removed a sufficient amount of batch variation to make it undetectable to our gPCA test statistic $\delta$. The mDWD method, however, did not remove sufficient batch variation to make it undetectable to $\delta$ in all but the GENEMAM scenario with run time considered as batch.

An analysis of the amount of time each of these methods took to run on the different data sets was also performed. Table 5.2 gives the run times for each of the analyses. On the $n = 90 \times p = 1000$ simulated data with only 2 batches and 2 phenotypes, all correction methods took less than 10 seconds to run in all cases. The filtered GENEMAM data (\(n = 77\))

\(^1\)http://www.openbioinformatics.org/penncvn/pennkv_tutorial_affy_gw6.html
$614 \times p = 1000$ with $b = 8$ plates) and the filtered GENOA data ($n = 703 \times p = 1000$ with $b = 10$ plates) with plate indicating batch took somewhat longer to run; however, mDWD took a prohibitively long time to run on this data (11.7 hours for the GENEMAM data with run time as batch, 144.4 hours for the GENEMAM data with plate as batch, and 495.6 hours for the GENOA data with plate as batch).

Table 5.1: Comparison of Batch Correction Methods: test statistic $\delta$ and corresponding $p$-values before and after batch correction for the three simulated data scenarios with no phenotypic effect, high variance phenotypic effect, and low variance phenotypic effect, and the two case study data sets, GENEMAM and GENOA. Batch correction methods used are (b) batch mean centering, (c) multiclass distance weighted discrimination, (d) non-parametric empirical Bayes, and (e) parametric empirical Bayes. Test results for the uncorrected data are given in column (a). A ‘NA’ indicates that that batch correction method was not possible for that data due to no phenotypic variable available.

<table>
<thead>
<tr>
<th></th>
<th>(a) Raw</th>
<th>(b) BMC</th>
<th>(c) mDWD</th>
<th>(d) EBn</th>
<th>(e) EBp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No Phenotype</strong></td>
<td>$0.901 &lt;0.001$</td>
<td>$0.068 1.000$</td>
<td>$0.621 0.815$</td>
<td>NA NA</td>
<td>NA NA</td>
</tr>
<tr>
<td><strong>High Variance Phenotype</strong></td>
<td>$0.794 &lt;0.001$</td>
<td>$0.033 1.000$</td>
<td>$0.617 0.008$</td>
<td>$0.323 1.000$</td>
<td>$0.311 1.000$</td>
</tr>
<tr>
<td><strong>Low Variance Phenotype</strong></td>
<td>$0.687 &lt;0.001$</td>
<td>$0.018 1.000$</td>
<td>$0.515 &lt;0.001$</td>
<td>$0.223 1.000$</td>
<td>$0.245 0.962$</td>
</tr>
<tr>
<td><strong>GENEMAM (run time)</strong></td>
<td>$0.583 &lt;0.001$</td>
<td>$0.044 1.000$</td>
<td>$0.450 0.056$</td>
<td>$0.227 1.000$</td>
<td>$0.226 1.000$</td>
</tr>
<tr>
<td><strong>GENEMAM (plate)</strong></td>
<td>$0.599 &lt;0.001$</td>
<td>$0.050 1.000$</td>
<td>$0.536 0.012$</td>
<td>$0.415 1.000$</td>
<td>$0.395 0.884$</td>
</tr>
<tr>
<td><strong>GENOA (plate)</strong></td>
<td>$0.922 &lt;0.001$</td>
<td>$0.017 1.000$</td>
<td>$0.893 &lt;0.001$</td>
<td>NA NA</td>
<td>NA NA</td>
</tr>
</tbody>
</table>

5.4.1 Batch Correction Sensitivity Analysis

An analysis of the sensitivity to batch correction was performed to compare the features found significant before and after batch correction. The simulated dataset as described above with dependent batch and phenotype were used. In our simulated dataset, there were 50 features with a phenotypic effect, 50 features with a batch effect, and 100 features with both a phenotypic and batch effect. The method of Benjamini and Hochberg [3] for adjusting for multiple testing was used at a significance level of $\alpha = 0.1$. 

78
Table 5.2: Run Time Analysis of Batch Correction Methods. Time is in seconds unless otherwise noted. Batch correction methods used are (a) batch mean centering, (b) multiclass distance weighted discrimination, (c) non-parametric empirical Bayes, and (d) parametric empirical Bayes. A ‘NA’ indicates that that batch correction method was not possible for that data due to no phenotypic variable available.

<table>
<thead>
<tr>
<th></th>
<th>(a) BMC</th>
<th>(b) mDWD</th>
<th>(c) EBn</th>
<th>(d) EBp</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Phenotype</td>
<td>0.016</td>
<td>3.790</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>High Variance Phenotype</td>
<td>0.019</td>
<td>4.931</td>
<td>8.428</td>
<td>0.121</td>
</tr>
<tr>
<td>Low Variance Phenotype</td>
<td>0.016</td>
<td>3.795</td>
<td>7.364</td>
<td>0.123</td>
</tr>
<tr>
<td>GENEMAM (run time)</td>
<td>0.034</td>
<td>11.737(^a)</td>
<td>19.445</td>
<td>0.335</td>
</tr>
<tr>
<td>GENEMAM (plate)</td>
<td>0.040</td>
<td>144.355(^a)</td>
<td>32.885</td>
<td>0.555</td>
</tr>
<tr>
<td>GENOA (plate)</td>
<td>0.049</td>
<td>495.581(^a)</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

\(^a\)Time in hours.

After fitting a linear model using the \texttt{lmFit()} function with phenotype as the predictor, the number of significant features in simulated data was assessed using the \texttt{eBayes()} function in the \texttt{limma} package both prior to batch correction and after batch correction using the empirical Bayes method of Johnson et al. [22] and the BMC method of Sims et al. [38]. For batch correction, the \texttt{Combat()} function in the \texttt{sca} package was used and both non-parametric and parametric empirical Bayes was implemented. BMC was also implemented using the \texttt{pamr.batchadjust()} function in the \texttt{pamr} package.

Using simulated data, we assessed the effects of correcting for batch on the number of significant features. Forty-eight of the 150 features had a significant phenotypic effect prior to batch correction while 148 of the 150 features were significant post-batch correction using BMC, 149 of the 150 features were significant post-batch correction using EBn, and 148 of the 150 features were significant post-batch correction using EBp (Table 5.3). This shows that batch correction allows features with a true phenotypic effect that is masked by batch to be identified as significant after batch correction.
Table 5.3: Contingency tables from simulated data with dependent batch and phenotypic effects that show the number of features truly significant versus those found to be significant using `lmFit()` and `eBayes()` on (a) raw data, (b) batch corrected data using batch mean-centering (BMC), (c) batch corrected data using non-parametric empirical Bayes (EBn), and (d) batch corrected data using parametric empirical Bayes (EBp). The rows of the tables indicate truth and the columns indicate the test results.

<table>
<thead>
<tr>
<th></th>
<th>Fail to reject</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Phenotype Effect</td>
<td>850</td>
<td>0</td>
</tr>
<tr>
<td>True Phenotype Effect</td>
<td>102</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Fail to reject</td>
<td>Reject</td>
</tr>
<tr>
<td>No Phenotype Effect</td>
<td>849</td>
<td>1</td>
</tr>
<tr>
<td>True Phenotype Effect</td>
<td>2</td>
<td>148</td>
</tr>
</tbody>
</table>

(a) Raw

<table>
<thead>
<tr>
<th></th>
<th>Fail to reject</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Phenotype Effect</td>
<td>848</td>
<td>2</td>
</tr>
<tr>
<td>True Phenotype Effect</td>
<td>1</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td>Fail to reject</td>
<td>Reject</td>
</tr>
<tr>
<td>No Phenotype Effect</td>
<td>848</td>
<td>2</td>
</tr>
<tr>
<td>True Phenotype Effect</td>
<td>2</td>
<td>148</td>
</tr>
</tbody>
</table>

(b) BMC Corrected

(c) EBn Corrected

(d) EBp Corrected

5.5 Discussion

Our gPCA $\delta$ statistic indicates that BMC and EB batch correction methods successfully remove the non-biological variation due to batch effects (all $p$-values > 0.05), but DWD does not in all cases. Computational time may make the mDWD method additionally unattractive given the performance of BMC and EB which mitigated the batch effects as desired in much less time. We note, also, that all data sets had large sample sizes, which is not always possible in microarray data. Lazar et al. [26] note that, of these four batch correction methods, only empirical Bayes does not require more than 25 samples to work correctly and that it can successfully remove batch effects with as few as 5 samples. Our sensitivity to batch correction analysis additionally shows that data with significant batch effects should be adjusted using a batch correction method prior to further analyses.
6 The gPCA Package for Identifying Batch Effects

Batch effects are commonly observed systematic non-biological variation between groups of samples due to experimental artifacts, such as processing date, lab, or technician. Combining samples from multiple batches can cause the true biological variation in a high-throughput experiment to be obscured by variation due to batch.

6.1 Guided Principal Components Analysis

Guided principal components analysis (gPCA) is an extension of principal components analysis (PCA) that replaces the data $X$ matrix in the singular value decomposition (SVD) of PCA with $Y'X$ such that

$$Y'X = UDV'$$

where $Y$ is an $n \times b$ indicator matrix where $n$ denotes sample and $b$ denotes batch. For $k = 1, \ldots, b$ batches, each is comprised of $n_k$ observations such that $\sum_{k=1}^{b} n_k = n$. The indicator matrix consists of $b$ blocks with $n_k$ rows for $k = 1, \ldots, b$, and $k$ columns where, for each block,

$$Y_k = \begin{cases} 1 & \text{if } k = b \\ 0 & \text{otherwise} \end{cases}$$

Performing SVD on $Y'X$ results in a $b \times b$ batch loadings matrix $U$ and a $p \times p$ probe loadings matrix $V$. Large singular values (the diagonal elements of the $q \times q$ matrix $D$ where $q = \min(n, p)$) imply that the batch is important for the corresponding principal component. gPCA guides the SVD to look for batch effects in the data based on the batch indicator matrix $Y$, which can be defined to indicate any type of potential batch effect, such as time of hybridization, plate, or other experimental artifact.
In Chapter 2, we proposed a test statistic $\delta$ that quantifies the proportion of variance due to batch effects in experimental genomic data. The proportion of total variance due to batch is taken to be the ratio of the variance of the first principal component from gPCA to the variance of the first principal component from unguided PCA

$$\delta = \frac{\text{var}(\mathbf{XV}_{g1})}{\text{var}(\mathbf{XV}_{u1})}$$

where $g$ indicates gPCA and $u$ indicates unguided PCA. $\mathbf{V}$ is the matrix of probe loadings resulting from gPCA or PCA, respectively. Large values of $\delta$ (values near 1) imply that the batch effect is large.

To determine whether $\delta$ is significantly larger than would be expected by chance, a $p$-value is estimated using a permutation distribution created by permuting the batch vector $M = 1000$ times so that $\delta_{pm}$ is computed for $m = 1, \ldots, M$ where $p$ indicates the permutation. Here $\delta_{pm}$ is the proportion of the total variance due to the first principal component from the $m^{th}$ permutation from gPCA to the total variance due to the first principal component from the $m^{th}$ permutation from unguided PCA. A one-sided $p$-value (testing $H_0 : \delta_{pm} = \delta$ versus $H_1 : \delta_{pm} > \delta$) is estimated as the proportion of times the observed $\delta$ was in the extreme tail of the permutation distribution

$$p\text{-value} = \frac{\sum_{m=1}^{M} (\delta_{pm} > \delta)}{M}.$$ 

For more details on gPCA see Chapter 2.
6.2 R Package

The gPCA package includes four example data sets, the gPCA.batchdetect() function that produces the $\delta$ statistic and corresponding $p$-value, and additional visualization functions.

6.2.1 Data

Four data sets are included in the gPCA package, three simulated data sets and one case study data set. The case study data (data(caseDat)) contains copy number variation data with $n = 500$ observations and $p = 1000$ features that were retained after a variance filter was applied.

The simulated data represents copy number data under three scenarios: (1) feature data (here, feature denotes probe) with no phenotypic variable (data(nopheDat)); (2) feature data with a high variance phenotypic variable (data(highpheDat)); and (3) feature data with a low variance phenotypic variable (data(lowpheDat)). The feature data were generated independently from a normal distribution with 1000 features and 90 observations. Data with two batches and two phenotypes were simulated. Batch means $\mu_{b_1} = 0$ and $\mu_{b_2} = 1$ and batch variance $\sigma_b = 0.5$ were used to simulate the data. The proportion of features affected by batch was $bprop = 0.01$ for the no phenotype scenario and $bprop = 0.05$ for the high and low variance phenotype scenarios.

For the scenarios with phenotypic effects, the proportion of features affected by phenotype was $pprop= 0.1$. The phenotypic means were $\mu_{p_1} = 0$ and $\mu_{p_2} = 1$ and the phenotypic variance was $\sigma_p = 2$ for the high variance phenotype scenario and $\sigma_p = 0.2$ for the low variance phenotype scenario. Chapter 3 provides an in depth description of the data simulations.

For all four data sets, the first column of the data frame containing the data contains the batch vector which indicates batch for the $n$ observations. The rest of the data frame
contains the uncentered feature data.

6.2.2 Application

The $\delta$ statistic, corresponding $p$-value from the permutation test, and various other measures are output by the `gPCA.batchdetect()` function. The syntax for this function is

```r
> out<-gPCA.batchdetect(x=data, batch=batch, center=FALSE,
+ filt=NULL, nperm=1000, seed=13)
```

where `x` is the $n \times p$ matrix of feature data `X`, `batch` is a length $n$ vector indicating batch which is used to calculate the `Y` matrix for gPCA. The option `center` is a logical indicating whether or not data is centered where `center=TRUE` if the data `x` is already centered. `nperm` indicates how many permutations will be used for calculating the permutation test statistic (defaults to 1000), `filt` gives the number of features to retain when applying a variance-based filter to the data (defaults to `NULL` indicating no filter applied), and `seed` sets `set.seed(seed)`.

Note that `x` must be complete data (i.e. contain no missing values) and the class of `x` must be "matrix". The function, when run actively, will ask if mean-value imputation should be performed for any missing values, but when run passively will cause an error.

The `gPCA.batchdetect()` function outputs the value of the statistic $\delta$, the associated $p$-value, the batch vector `batch`, the $M$ values of $\delta_p$ resulting from the permutation test, the proportion of variance associated with the first principal component from unguided (PCu) and guided (PCg) PCA, as well as the cumulative variance associated with all $n$ principal components resulting from unguided PCA (`cumulative.var.x`) and the cumulative variance associated with all $b$ principal components resulting from gPCA (`cumulative.var.g`).

The `gPCA` package also has three functions to visualize the data. The function `gDist` produces a density plot of the $\delta_p$ values output by the `gPCA.batchdetect` function. The
function \texttt{PCplot} produces principal component plots of either the unguided or guided principal components and allows for either directly comparing the first two principal components, or comparing the first \texttt{npcs} principal components. Finally, the function \texttt{CumulativeVarPlot} produces a plot of the cumulative variance from guided or unguided PCA.

\begin{verbatim}
> gDist(out)
> PCplot(out,ug="guided",type="1v2")
> PCplot(out,ug="guided",type="comp",npcs=3)
> CumulativeVarPlot(out,ug="unguided",col="blue")
\end{verbatim}

6.3 Example

We will discuss a brief example using \texttt{caseDat} data from the \texttt{gPCA} package. We first load the data \texttt{caseDat} and assign the first column to \texttt{batch}. The rest of the data frame is the feature data, so we assign that to \texttt{dat} and re-classify it as a matrix. Since the \texttt{caseDat} feature data is already centered, we set \texttt{center=TRUE}. The value of the test statistic $\delta$ and the corresponding $p$-value are easily printed and the percent of total variation that is explained by batch is calculated.

\begin{verbatim}
> data(caseDat)
> batch<-caseDat$batch
> dat<-as.matrix(caseDat[,,-1])
> out<-gPCA.batchdetect(x=dat,batch=batch,center=TRUE)
> out$delta ; out$p.val

[1] 0.5529794
[1] "<0.001"

> ((out$varPCg1-out$varPCu1/out$varPCg1)*100

[1] 96.2252
\end{verbatim}
Figure 6.1: Distribution plot of $\delta_p$ values

We can also plot the distribution of the $\delta_p$ values from the permutation test and see where our test statistic $\delta$ (represented by the red dashed line) falls in comparison (Figure 6.1). Plots of the first versus the second principal components from gPCA can be plotted (Figure 6.2) as well as a sample of the first few principal comparisons (Figure 6.3).
> par(mai=c(0.8,0.8,0.1,0.1),cex=0.8)
> PCplot(out,ug="guided",type="1v2")

Figure 6.2: Principal components plot of first two principal components from gPCA

6.4 Conclusion

The gPCA package provides functionality to test for batch effects in high-throughput genomic data using the function `gPCA.batchdetect()`. The ability to detect batch effects in genomic
data allows further batch correction procedures such as batch mean-centering [38], distance weighted discrimination (DWD) [2, 17, 18, 32], or empirical Bayes [22], to be employed to attempt to remove the unwanted variation due to batch effects. However, correcting for
batch when there is no significant batch effect may result in removing biological variation instead of the systematic non-biological variation due to batch. This package provides the ability to perform a test to detect batch effects.

6.5 Session Info

> sessionInfo()

R version 2.15.2 (2012-10-26)
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

locale:

attached base packages:
[1] stats graphics grDevices utils datasets methods base

other attached packages:
[1] gPCA_1.0

loaded via a namespace (and not attached):
[1] tools_2.15.2
7 Discussion

Non-experimental variation due to the occurrence of batch effects in high-throughput genomic data is a common problem that can have a serious impact on statistical testing and on conclusions made by high-throughput experiments. Guided principal components analysis can be used to test for batch effects in large and messy data such as expression data, CNV data, methylation data, etc., by computing the SVD while taking batch into account. Principal components plots are a standard method of looking for batch effects in high-throughput data. A gPCA plot allows an investigator to specifically identify batch effects that are potentially hidden in a PCA plot. For example a PCA plot might not show any difference between batches if batch is not the largest source of variation, however, gPCA plots will. The PCA and gPCA plots of the GENEMAM data both show the batch separation because batch is the largest source of variation in this data. The PCA plot of the GENOA data does not show any real separation in the data, however the gPCA plot shows that plate 4 separates from the rest of the data indicating that there is a significant batch effect. In both instances, further quantitative analyses are used to confirm that there is a significant batch effect.

The Y matrix in the gPCA analysis can be formed by considering any combination of variables. We do note that with the Y matrix coding multiple variables, the variance ascribed to the first principal component of the gPCA may incorporate multiple sources, which would be difficult to disentangle. To estimate the variance attributed to multiple sources, gPCA could be used to examine each one by defining Y in separate analyses. Note that gPCA is dependent on knowing how to define potential batch effects. If this is not known then the gPCA δ statistic can not be used. If batch is misspecified by the investigator, provided the
misspecified batch effect indicator matrix has no relationship to the experimental design, then the test will likely not reject the null hypothesis because Type I error was close at the nominal 0.05 level.

From our simulation studies in Chapter 3, the Type I error of our statistic is close to nominal 0.05 level and power is reasonably good when an adequate proportion of the features are affected by batch. A simulation analysis when the proportion of features affected by batch was high (between 50 and 90%) was also performed and we found that the estimated power was 100% (Table 3.4). We also note that features were simulated as independent in our simulation studies. Any correlation between features in case study data is likely due to pathways and not probe design which affects batch.

In Chapter 4, we applied our statistic to two different sets of copy number variation data, one with obvious and known batch effects (GENEMAM) and one with less obvious batch effects (GENOA). In both we were able to use gPCA to determine plates with large variation from the other plates. gPCA in the GENEMAM data allowed us to see that plate 3 had potential quality issues which we investigated further by looking at heatmaps of PC\(_1\) and PC\(_2\) (Figure 4.2). Based on these we saw that plate 3 did indeed have a quality problem that was associated with sample well location on the plate. For the GENOA data, gPCA showed that there was a significant batch effect that identified plate 4 as a batch that unguided PCA did not recognize since batch effects did not dominate the variance. For both sets of data, our \(\delta\) statistic was highly significant indicating the presence of batch effects.

Scaling of data is a common statistical practice prior to data analysis; however, in the case of microarray data, scaling of the batch identifier matrix \(Y\) by batch sample size \(n_k\) for each batch \(k\) is not in general useful for balanced experiments. However, when some
batches have far more samples than others, scaling of $Y$ is a useful tool to correct for the imbalance. In the case of the GENEMAM data, while plates 5 and 8 had half as many or fewer samples than the rest of the plates, the effect of scaling $Y$ was minimal ($\delta = 0.5576; p$-value < 0.001), though it did have an effect on the $\delta$ statistic, but not on the significance of the batch effects. Simulation analyses varying the sizes of the batches found no difference between scaling $Y$ by sample size or not (Table 3.6). For microarray data, we do not want to scale the data matrix $X$ since all the variables, probes in our case, are already on the same scale and scaling $X$ would only serve to adjust the variance. If the variances are smoothed then we may miss an important difference between variables or batches.

gPCA can be used on other problems and types of data as well, including B-allele frequency data, expression data, and RNA-Seq data. Since preprocessing of microarrays is time consuming, expensive, and with abundant systematic errors, the ability to discover and adjust for these errors is important. Our test statistic $\delta$ that employs gPCA allows one to test for significant sources of systematic errors, or batch effects, in all types of high-throughput data.

After detecting a batch effect in high-throughput data, the non-experimental variation due to batch effects must be adjusted for prior to further analysis. Table 1.1 lists various methods for adjusting for batch effect in analysis and these methods were also discussed in Chapter 1. However, they do not incorporate a procedure for identifying whether a batch effect is truly present. The table also provides articles that have implemented these methods and corresponding data types on which they have been implemented. Using both simulated and real data (see Chapter 5), we further assessed the effects of correcting for batch on the number of significant features. In our simulated dataset, there were 50 features with a
phenotypic effect, 50 features with a batch effect, and 100 features with both a phenotypic and batch effect. After fitting a linear model using the \texttt{lmFit()} function with phenotype as the predictor, the number of significant features in simulated data was assessed using the \texttt{eBayes()} function in the \texttt{limma} package both prior to batch correction and after batch correction using the batch mean-centering method of Sims et al. [38] and the empirical Bayes method of Johnson et al. [22] and the method of Benjamini and Hochberg [3] for adjusting for multiple testing letting $\alpha = 0.1$. Forty-eight of the 150 features had a significant phenotypic effect prior to batch correction while 148 of the 150 features were significant post-batch correction (Table 5.3). This shows that batch correction allows features with a true phenotypic effect that is masked by batch to be identified as significant after batch correction.

Luo et al. [31] looked at the impact of batch effect removal on cross-batch prediction performance and Lazar et al. [26] and Chen et al. [8] provide surveys of some of the many methods of batch effect removal. Our proposed test statistic is useful for identifying whether any of the listed batch adjustment methods should be applied prior to statistical analysis and, after batch correction by a given method, whether that method successfully adjusted the data for batch effects. In Chapter 5 we applied four methods from Table 1.1, batch-mean centering (BMC), distance weighted discrimination (DWD), and both non-parametric and parametric versions of empirical Bayes (EBn and EBp), to three simulated data sets and the variance filtered (to 1000 features) GENEMAM and GENOA data sets. All five data sets had significant batch effects prior to batch correction, and we found that post-batch correction, BMC and the empirical Bayes methods successfully adjusted the data for batch effects, while DWD did not. How well each batch effect correction method works, in terms of retaining the biological variation of the experiment, and the types of data for which each
method is best suited, have been previously addressed by several articles [8, 26, 31] and is not in the scope of this paper.

We have provided an R package (discussed in Chapter 6) that provides functions that perform gPCA and plot various visualizations of the results, as well as example data sets. This package should make testing for the existence of batch effects in high-throughput genomic data considerably easier since it provides publicly available, user-friendly functions to test for batch effects using the gPCA $\delta$ statistic and to plot corresponding PCA plots and $\delta$ density plots.

The research presented in this thesis brought up several additional lines of research. Potential topics for future work include application of our statistic to other data types such as RNA sequencing data, development of a new batch correction method based on gPCA, and extension of gPCA into other types of analyses such as toxicological dose-response/exposure data as $X$ with mixture group indicated by $Y$ instead of batch.
Bibliography


A Appendix

A.1 R Code

This appendix contains pertinent R code from the analyses in this dissertation.

A.1.1 Chapter 1: Introduction

The following syntax reproduces Figure 1.1

```r
> ## Full GENEMAM data (mean-value imputed)
> ## Apply Pearson's Correlation as distance measure &
> ## perform hierarchical clustering
> rho.x<-cor(t(x2.imp))
> dist.mx.x<-as.dist(1-rho.x)
> out.hclust.x<-hclust(dist.mx.x,method="average")
> out.hclust.x$labels<-batch
> plot(out.hclust.x,main="",xlab=expression(1-rho))
```

A.1.2 Chapter 3: Simulation Study

Three functions were used to produce the no phenotype, high variance phenotype, and low variance phenotype data and perform the permutation test, which was then repeated to estimate Type I error and Power. The syntax of the function to perform the no phenotype simulation is

```r
> no.phe

function (n = 90, p = 1000, b = 2, s, bprop, beffect, scenario,
         niter = 500, nperm = 1000, error = FALSE, plotout, plots = FALSE)
{
  casename <- "nophe"
  plotcase <- "No Phenotype"
  set.seed(13)
  batch <- rep(1:b, each = n/b)
  batch <- sample(batch, replace = FALSE)
```

99
permute <- matrix(NA, ncol = length(batch), nrow = 50000)
for (j in 1:50000) {
    permute[j, ] <- sample(batch, replace = FALSE)
}
p.val <- numeric()
for (k in 1:niter) {
    print(paste("k=", k, sep = ")
    samp <- sample(1:dim(permute)[1], nperm, replace = FALSE)
    permute.samp <- permute[samp, ]
    if (error == TRUE) {
        x <- rmvnorm(n, mean = rep(0, p), sigma = diag(s, nrow = p))
    } else {
        bats <- array(dim = c(n/b, p, b))
        bats[, , 1] <- rmvnorm(n/b, mean = rep(beffect[1], p), sigma = diag(s, nrow = p))
        for (i in 2:b) {
            bats[, , i] <- rmvnorm(n/b, mean = c(rep(beffect[1], p - (bprop * p)), rep(beffect[i], (bprop * p))), sigma = diag(s, nrow = p))
        }
        bat <- data.frame(matrix(nrow = n, ncol = p))
        for (i in 1:b) {
            bat[which(batch == i), ] <- bats[, , i]
        }
        x <- bat
    }
    x2 <- scale(x, scale = F)
    svd.out <- svd(x2)
    var.x <- var(x2 %% svd.out$v)
    total.unguided.var <- sum(diag(var.x))
    PC.u <- diag(var.x)[1]/sum(diag(var.x))
    y.bat <- matrix(0, n, b)
    for (j in 1:b) {
        y.bat[, j] <- ifelse(batch == j, 1, 0)
    }
    y2.bat <- scale(y.bat, scale = F)
gsvd.out.bat <- svd(t(y2.bat) %*% x2)
var.x.bat <- var(x2 %*% gsvd.out.bat$v)
total.guided.batch <- sum(diag(var.x.bat))
PC.g <- diag(var.x.bat)[1]/sum(diag(var.x.bat))
delta <- diag(var.x.bat)[1]/diag(var.x)[1]

cp.plot <- function(pc1.x, pc2.x, pc1.bat, pc2.bat) {
    outplot <- paste(plotout, "PCplots/", sep = "")
    png(file = paste(outplot, casename, "/scenario", "/PCplots_{k}.png", sep = ""),
         height = 500, width = 1000)
    par(mfrow = c(1, 2))
    plot(pc1.x, pc2.x, col = c("red", "blue")[batch],
         main = "Unguided", xlab = "PC1", ylab = "PC2")
    legend(x = "bottom", legend = c("b1", "b2"), col = c("red", "blue")[unique(batch)], inset = 0.03)
    plot(pc1.bat, pc2.bat, col = c("red", "blue")[batch],
         main = "Y=batch", xlab = "PC1", ylab = "PC2")
    legend(x = "bottom", legend = c("b1", "b2"), col = c("red", "blue")[unique(batch)], inset = 0.03)
    dev.off()
}
if (plots == TRUE)
    pc.plot(x2 %*% svd.out$v[, 1], x2 %*% svd.out$v[, 2], x2 %*% gsvd.out.bat$v[, 1], x2 %*% gsvd.out.bat$v[, 2])

delta.p <- numeric()
for (i in 1:nperm) {
    batch.p <- permute.samp[i, ]
y.bat.p <- matrix(0, n, b)
    for (j in 1:b) {
        y.bat.p[, j] <- ifelse(batch.p == j, 1, 0)
    }
y2.bat.p <- scale(y.bat.p, scale = F)
gsvd.out.bat.p <- svd(t(y2.bat.p) %*% x2)
var.x.bat.p <- var(x2 %*% gsvd.out.bat.p$v)
total.guided.batch.p <- sum(diag(var.x.bat.p))
PC.g.p <- diag(var.x.bat.p)[1]/sum(diag(var.x.bat.p))
delta.p[i] <- diag(var.x.bat.p)[1]/diag(var.x)[1]
p.val[k] <- sum(delta < delta.p)/length(delta.p)
dens.plot <- function(stat, stat.p) {
  outplot2 <- paste(plotout, "DensityPlots/", sep = "")
  png(file = paste(outplot2, casename, "/scenario",
                 scenario, "/DensityPlot_", k, ".png", sep = "") ,
      height = 500, width = 500)
  plot(density(stat.p), main = paste("Distribution of Delta for a ",
                 plotcase, "\n(Scenario", scenario, "; delta=",
                 round(stat, 3), "; p-value=" , round(p.val[k],
                 3), ")", sep = ""), xlim = c(min(stat.p, stat),
                 max(stat.p, stat)))
  abline(v = stat, col = "red")
  dev.off()
}
if (plots == TRUE) {
  dens.plot(delta, delta.p)
}
prop <- sum(p.val < 0.05)/length(p.val)
if (error == FALSE) {
  mat <- data.frame(s, bprop, prop)
  rownames(mat) <- paste("Scenario", scenario)
  list(mat = mat, sigma = s, bprop = bprop, beffect = beffect,
       prop = prop, p.val = p.val, delta = delta, delta.p = delta.p,
       batch = batch)
}
else {
  mat <- data.frame(s, "-", prop)
  rownames(mat) <- paste("Scenario", scenario)
  list(mat = mat, sigma = s, prop = prop, p.val = p.val,
       delta = delta, delta.p = delta.p, batch = batch)
}

The syntax of the function to perform simulations with a true phenotype is

> di.phe
function (n = 90, p = 1000, b = 2, nphe = 2, s, sp = 2, bprop,
beffect, pprop = 0.1, peffect = c(0, 2), scenario, niter = 500,
nperm = 1000, error = FALSE, plots = FALSE, plotout)
{
casename <- "diphe"
plotcase <- "Dichotomous Phenotype"
set.seed(13)
batch <- rep(1:b, each = n/b)
batch <- sample(batch, replace = FALSE)
pheno <- rep(1:nphe, each = n/nphe)
permute <- matrix(NA, ncol = length(batch), nrow = 50000)
for (j in 1:50000) {
    permute[j, ] <- sample(batch, replace = FALSE)
}
p.val <- numeric()
for (k in 1:niter) {
    print(paste("k=", k, sep = ")
    samp <- sample(1:dim(permute)[1], nperm, replace = FALSE)
    permute.samp <- permute[samp, ]
    phes <- array(dim = c(n/nphe, p, nphe))
    phes[, , 1] <- rmvnorm(n/nphe, mean = rep(peffect[1], p),
                        sigma = diag(sp, nrow = p))
    for (i in 2:nphe) {
        phes[, , i] <- rmvnorm(n/nphe, mean = c(rep(peffect[i],
                                                  pprop * p),
                                                  rep(peffect[1], p - pprop * p)),
                        sigma = diag(sp, nrow = p))
    }
    phe <- data.frame(matrix(nrow = n, ncol = p))
    for (i in 1:nphe) {
        phe[which(pheno == i), ] <- phes[, , i]
    }
    if (error == TRUE) {
        x <- phe
    } else {
        bats <- array(dim = c(n/b, p, b))
        bats[, , 1] <- rmvnorm(n/b, mean = rep(beffect[1], p),
                              sigma = diag(s, nrow = p))
        for (i in 2:b) {
    }
bats[, , i] <- rmvnorm(n/b, mean = c(rep(beffect[1],
p - (bprop * p)), rep(beffect[i], (bprop * 
p))), sigma = diag(s, nrow = p))
}
bat <- data.frame(matrix(nrow = n, ncol = p))
for (i in 1:b) {
    bat[which(batch == i), ] <- bats[, , i]
}
x <- phe + bat
}
x2 <- scale(x, scale = F)
svd.out <- svd(x2)
var.x <- var(x2 %*% svd.out$v)
total.guided <- sum(diag(var.x))
PC.u <- diag(var.x)[1]/sum(diag(var.x))
y.bat <- matrix(0, n, b)
for (j in 1:b) {
    y.bat[, j] <- ifelse(batch == j, 1, 0)
}
y2.bat <- scale(y.bat, scale = F)
gsvd.out.bat <- svd(t(y2.bat) %*% x2)
var.x.bat <- var(x2 %*% gsvd.out.bat$v)
total.guided.batch <- sum(diag(var.x.bat))
PC.g <- diag(var.x.bat)[1]/sum(diag(var.x.bat))
delta <- diag(var.x.bat)[1]/diag(var.x)[1]
 pc.plot <- function(pc1.x, pc2.x, pc1.bat, pc2.bat) {
    outplot <- paste(plotout, "PCplots/", sep = "")
    png(file = paste(outplot, casename, "/scenario", "/PCplots_{", k, ",png", sep = ""),
        height = 500, width = 1000)
    par(mfrow = c(1, 2))
    plot(pc1.x, pc2.x, col = c("red", "blue")[batch],
        pch = pheno, main = "Unguided", xlab = "PC1",
        ylab = "PC2")
    legend(x = "bottom", legend = c("b1p1", "b2p1", "b1p2",
        "b2p2"), col = c("red", "blue")[unique(batch)],
        pch = rep(1:2, each = 2), inset = 0.03)
    plot(pc1.bat, pc2.bat, col = c("red", "blue")[batch],
        pch = pheno, main = "Guided", xlab = "PC1",
        ylab = "PC2")
    legend(x = "bottom", legend = c("b1p1", "b2p1", "b1p2",
        "b2p2"), col = c("red", "blue")[unique(batch)],
        pch = rep(1:2, each = 2), inset = 0.03)
}
pch = pheno, main = "Y=batch", xlab = "PC1",
ylab = "PC2")
legend(x = "bottom", legend = c("b1p1", "b2p1", "b1p2",
"b2p2"), col = c("red", "blue") [unique(batch)],
pch = rep(1:2, each = 2), inset = 0.03)
dev.off()
}
if (plots == TRUE)
pc.plot(x2 %*% svd.out$v[, 1], x2 %*% svd.out$v[, 2], x2 %*% gsvd.out.bat$v[, 1], x2 %*% gsvd.out.bat$v[, 2])
delta.p <- numeric()
for (i in 1:nperm) {
  batch.p <- permute.samp[i, ]
y.bat.p <- matrix(0, n, b)
  for (j in 1:b) {
    y.bat.p[, j] <- ifelse(batch.p == j, 1, 0)
  }
y2.bat.p <- scale(y.bat.p, scale = F)
gsvd.out.bat.p <- svd(t(y2.bat.p) %*% x2)
var.x.bat.p <- var(x2 %*% gsvd.out.bat.p$v)
total.guided.batch.p <- sum(diag(var.x.bat.p))
PC.g.p <- diag(var.x.bat.p)[1]/sum(diag(var.x.bat.p))
delta.p[i] <- diag(var.x.bat.p)[1]/diag(var.x)[1]
}
p.val[k] <- sum(delta < delta.p)/length(delta.p)
dens.plot <- function(stat, stat.p) {
  outplot2 <- paste(plotout, "DensityPlots/", sep = "")
  png(file = paste(outplot2, casename, "/scenario",
    scenario, "/DensityPlot_", k, ".png", sep = ""),
    height = 500, width = 500)
  plot(density(stat.p), main = paste("Distribution of Delta for a ",
    plotcase, ">
    n(Scenario", scenario, "; delta=",
    round(stat, 3), "; p-value="), round(p.val[k],
    3), ")", sep = ""), xlim = c(min(stat.p, stat),
    max(stat.p, stat)))
  abline(v = stat, col = "red")
  dev.off()
if (plots == TRUE) {
    dens.plot(delta, delta.p)
}

prop <- sum(p.val < 0.05)/length(p.val)
if (error == FALSE) {
    mat <- data.frame(s, sp, pprop, bprop, prop)
    rownames(mat) <- paste("Scenario", scenario)
    list(mat = mat, s = s, sp = sp, bprop = bprop, beffect = beffect,
         peffect = peffect, pprop = pprop, prop = prop, p.val = p.val,
         delta = delta, delta.p = delta.p, batch = batch,
         pheno = pheno)
} else {
    mat <- data.frame(s, sp, pprop, "-", prop)
    rownames(mat) <- paste("Scenario", scenario)
    list(mat = mat, s = s, sp = sp, pprop = pprop, peffect = peffect,
         pprop = pprop, prop = prop, p.val = p.val, delta = delta,
         delta.p = delta.p, batch = batch, pheno = pheno)
}

Example calls of these functions is

> out.nopheno<-no.phe(n=90,p=1000,b=2,s=0.5,bprop=0.01,beffect=c(0,1),
+   scenario=1,niter=500,nperm=1000,error=FALSE,plots=FALSE)
> ##
> out.highpheno<-di.phe(n=90,p=1000,b=2,nphe=2,s=0.5,sp=2,bprop=0.01,
+   beffect=c(0,1),scenario=2,niter=500,nperm=1000,error=FALSE,plots=FALSE)
> ##
> out.lowpheno<-di.phe(n=90,p=1000,b=2,nphe=2,s=0.5,sp=0.2,bprop=0.01,
+   beffect=c(0,1),scenario=3,niter=500,nperm=1000,error=FALSE,plots=FALSE)

These calls would be repeated using the various parameters listed in Tables 3.1 and 3.2.

The following sim.data() function is used to simulate data for any of the three scenarios.

It produces a single data set that can be used in further analyses.
> sim.data

function (grid, g, b = 2, s = 0.5, bprop, beffect = c(0, 1), nphe = NULL, sp = NULL, pprop = NULL, peffect = NULL)
{
    n = grid$n[g]
    p = grid$p[g]
    pheno.genes = NULL
    if (length(beffect) != b) {
        stop("Number of batch effect means does not equal the number of batches."")
    }
    batch <- rep(1:b, each = n/b)
    batch <- sample(batch, replace = FALSE)
    pheno <- rep(1:nphe, each = n/nphe)
    print("Computing bat matrix.")
    bats <- list()
    bats[[1]] <- rmvnorm(n/b, mean = rep(beffect[1], p), sigma = diag(s, nrow = p))
    for (i in 2:b) {
        bats[[i]] <- rmvnorm(n/b, mean = c(rep(beffect[1], p - (bprop * p)), rep(beffect[i], (bprop * p))), sigma = diag(s, nrow = p))
    }
    bat <- data.frame(matrix(nrow = n, ncol = p))
    for (i in 1:b) {
        bat[which(batch == i), ] <- bats[[i]]
    }
    x <- bat
    print("Computing phe matrix.")
    phes <- list()
    phes[[1]] <- rmvnorm(n/nphe, mean = rep(peffect[1], p), sigma = diag(sp, nrow = p))
    for (i in 2:nphe) {
        phes[[i]] <- rmvnorm(n/nphe, mean = c(rep(peffect[i], pprop * p), rep(peffect[1], p - pprop * p)), sigma = diag(sp, nrow = p))
    }
    phe <- data.frame(matrix(nrow = n, ncol = p))
    for (i in 1:nphe) {
phe[which(pheno == i), ] <- phes[[i]]
}
x <- x + phe
print("Scaling X matrix.")
x2 <- scale(x, scale = F)
list(batch = batch, x = x, x2 = x2, n = n, p = p, b = b,
s = s, bprop = bprop, beffect = beffect, nphe = nphe,
sp = sp, pprop = pprop, peffect = peffect, pheno = pheno,
phes = phes, pheno.genes = pheno.genes)
}

Example calls of this function to produce data with a high variance phenotypic effect is

```r
> grid<-expand.grid(n=90,p=1000)
> data<-sim.data(grid=grid,g=1,b=2,s=0.5,bprop=0.05,beffect=c(0,1),
+ nphe=2,sp=2,pprop=0.1,peffect=c(0,1))


> names(data)

[1] "batch"   "x"    "x2"   "n"     "p"
[6] "b"       "s"    "bprop" "beffect" "nphe"
[11] "sp"      "pprop" "peffect" "pheno"  "phes"
[16] "pheno.genes"

> dim(data$x)

[1] 90 1000
```

This call can be repeated using the parameters listed in Tables 3.1 and 3.2 to produce simulated data sets.

To perform gPCA and produce our statistic δ, the function `gPCA.batchdetect()` was used. This function is also found in our R package that is discussed in Chapter 6.
> gPCA.batchdetect
function (x, batch, filt = NULL, nperm = 1000, center = FALSE, scaleY = FALSE, seed = 13)
{
    set.seed(seed)
    permute <- matrix(NA, ncol = length(batch), nrow = 50000)
    for (j in 1:50000) {
        permute[j, ] <- sample(batch, replace = FALSE)
    }
    samp <- sample(1:dim(permute)[1], nperm, replace = FALSE)
    permute.samp <- permute[samp, ]
    if (center == FALSE) {
        x2 <- scale(x, center = T, scale = F)
    } else {
        x2 <- x
    }
    if (sum(is.na(x)) > 0) {
        missing <- readline(prompt = "Missing values detected. Continue
            with mean value imputation? (Note this may take a very
            long time, but it will automatically save in your working
            dir so you don't have to ever run it again.) [y/n] ")
        if (substr(missing, 1, 1) == "n") {
            stop("The PC cannot be calculated with missing values."")
        } else {
            x2.imp <- ifelse(is.na(x2), rowMeans(x2, na.rm = TRUE), x2)
            save(x2.imp, "x2.imputed.RData")
        }
    } else {
        x2.imp <- x2
    }
    if (is.null(filt)) {
        data.imp <- x2.imp
    } else {

sd <- apply(x2.imp, 2, sd)
rank <- rank(sd)
keep <- (1:length(sd))[rank %in% (length(rank) - filt + 1):length(rank)]
data.imp <- x2.imp[, keep]

n <- dim(data.imp)[1]
p <- dim(data.imp)[2]
b <- length(unique(batch))
n
p
b
if (length(batch) != n) {
  stop("Matrices do not conform: length(batch)!n")
}
y <- matrix(nrow = length(batch), ncol = length(unique(batch)))
for (j in 1:length(unique(batch))) {
  y[, j] <- ifelse(batch == j, 1, 0)
}
if (scaleY == FALSE) {
  y2 <- scale(y, center = T, scale = F)
} else {
  ys <- matrix(nrow = length(batch), ncol = length(unique(batch)))
  nk <- apply(y, 2, sum)
  for (j in 1:length(unique(batch))) {
    ys[, j] <- ifelse(batch == j, 1/nk[j], 0)
  }
  y2 <- scale(ys, center = F, scale = F)
}

svd.x <- svd(data.imp)
PC.u <- data.imp %*% svd.x$v
var.x <- var(PC.u)
varPCu1 <- diag(var.x)[1]/sum(diag(var.x))
cumulative.var.u <- numeric()
for (i in 1:dim(var.x)[1]) {
  cumulative.var.u[i] <- sum(diag(var.x)[1:i])/sum(diag(var.x))
}
svd.bat <- svd(t(y2) %*% data.imp)
PC.g <- data.imp %*% svd.bat$v
var.bat <- var(PC.g)
varPCg1 <- diag(var.bat)[1]/sum(diag(var.bat))
cumulative.var.g <- numeric()
for (i in 1:dim(var.bat)[1]) {
    cumulative.var.g[i] <- sum(diag(var.bat)[1:i])/sum(diag(var.bat))
}
delta <- diag(var.bat)[1]/diag(var.x)[1]
delta.p <- numeric()
for (i in 1:nperm) {
    batch.p <- permute.samp[i, ]
y <- ys <- matrix(nrow = length(batch.p), ncol = length(unique(batch.p)))
    for (j in 1:length(unique(batch.p))) {
        y[, j] <- ifelse(batch.p == j, 1, 0)
    }
    if (scaleY == FALSE) {
        y2 <- scale(y, center = T, scale = F)
    } else {
        nk <- apply(y, 2, sum)
        for (j in 1:length(unique(batch.p))) {
            ys[, j] <- ifelse(batch.p == j, 1/nk[j], 0)
        }
        y2 <- scale(ys, center = F, scale = F)
    }
    svd.bat.p <- svd(t(y2) %*% data.imp)
    var.bat.p <- var(data.imp %*% svd.bat.p$v)
    PC.g.p <- diag(var.bat.p)[1]/sum(diag(var.bat.p))
    delta.p[i] <- diag(var.bat.p)[1]/diag(var.x)[1]
}
p.val <- sum(delta < delta.p)/length(delta.p)
p.val
p.val <- ifelse(p.val == 0, "<0.001", round(p.val, 3))
out <- list(delta = delta, p.val = p.val, delta.p = delta.p,
    batch = batch, filt = filt, n = n, p = p, b = b, PCg = PC.g,
    PCu = PC.u, varPCu1 = varPCu1, varPCg1 = varPCg1, nperm = nperm,
    cumulative.var.u = cumulative.var.u, cumulative.var.g = cumulative.var.g)
If missing values are detected in the data matrix, R will print the prompt “Missing values detected. Continue with mean value imputation? (Note this may take a very long time, but it will automatically save in your working dir so you don't have to ever run it again.) [y/n]”. If R is being run interactively, then the user has the ability to input “y” or “’n” for ‘yes’ or ‘no’, otherwise, an error occurs. An example call to this function is

```r
> out<-gPCA.batchdetect(x=data$x,batch=data$batch,center=FALSE)
> out$delta ; out$p.val

[1] 0.753347
[1] "<0.001"
```

where `data` is the high variance phenotype data simulated above.

The following code reproduces our sensitivity to filtering analysis for the low variance phenotype data

```r
> grid<-expand.grid(n=90,p=20000)
> filter<-c(10,100,1000,2000,5000,10000,15000)
> time.sim<-system.time(
+ SimDat<-sim.data(grid,g=1,b=2,s=0.5,bprop=0.03,beffect=c(0,1),
+ nphe=2,sp=0.2,pprop=0.1,peffect=c(0,1))
+ )
> sys.time<-pval<-deltav<-nfeat<-numeric()
> for (i in filter){
+     print(paste('filt=',i,sep=""))
+     times<-system.time(
+         out<-gPCA(x=SimDat$x2,batch=SimDat$batch,filt=i,center=TRUE)
+     )
+     save(out,times,file=paste(outfile,"lowpheSensData",i,".RData",sep=""))
+     sys.time<-c(sys.time,times[[3]])
+     pval<-c(pval,out$'p.val')
+     deltav<-c(deltav,out$delta)
+     nfeat<-c(nfeat,out$p)
+     print(paste('sys.time=',times[[3]],sep=""))
```

112
times<-system.time(
    out<-gPCA(x=SimDat$x2,batch=SimDat$batch,filt=NULL,nperm=1000,
               center=TRUE)
)

sys.time<-c(sys.time,times[[3]])
pval<-c(pval,out$p.val)
deltav<-c(deltav,out$delta)
nfeat<-c(nfeat,out$p)

### A.1.3 Chapter 4: Applications

The GENEMAM data set was mean-value imputed and all further analyses used the imputed data. The syntax used to impute the GENEMAM data (or any data set) is

```r
> data.imp<-ifelse(is.na(data),rowMeans(data,na.rm=TRUE),data)
```

gPCA was performed on the GENEMAM and GENOA data sets as described previously using the `gPCA.batchdetect()` function. The following syntax reproduces the principal component plot in Figure 4.2b where `PC.bat` and `PC.x` are the principal components matrices resulting from guided and unguided PCA on the GENEMAM data. Similar syntax was used to produce the PCA plots in Figures 4.2a and 4.4a and the gPCA plot in Figure 4.4b.

```r
> colors<-c("blue1","firebrick","darkorchid","aquamarine4","coral1",
             "deeppink","green4","gold")
>
> PC1lim<-c(min(PC.x[,1],PC.bat[,1]),max(PC.x[,1],PC.bat[,1]))
> PC2lim<-c(min(PC.x[,2],PC.bat[,2]),max(PC.x[,2],PC.bat[,2]))
> par(mai=c(0.65,0.65,0.1,0.1),cex=0.8)
> plot(PC.bat[,1],PC.bat[,2],pch=c(1:8)[batch],col=colors[batch],
      xlab=expression(PC[1]),ylab=expression(PC[2]),xlim=PC1lim,ylim=PC2lim)
> legend(x="bottom",legend=paste("Plate",1:8),ncol=4,col=colors,
        pch=1:8,inset=0.03)
```

The syntax to produce the heatmaps in Figure 4.2 using the `lattice` package is
> library(lattice)
> pc1.x<-PC.x[,1]
> pc2.x<-PC.x[,2]
> well.pos<-function(plate){
+   well<-as.character(demo$sample.well[demo$plate==plate])
+   wellspl<-unlist(strsplit(well,split=""))
+   let<-wellspl[seq(1,length(wellspl),by=3)]
+   num.mx<-matrix(wellspl[seq(1,length(wellspl),by=3)],ncol=2,byrow=T)
+   num<-paste(num.mx[,1],num.mx[,2],sep="")
+   out<-data.frame(well,let,num=as.numeric(num),
+                   pc1=pc1.x[demo$plate==plate],pc2=pc2.x[demo$plate==plate])
+   out
+ }
> plate1<-well.pos(1)
> plate2<-well.pos(2)
> plate3<-well.pos(3)
> plate4<-well.pos(4)
> plate5<-well.pos(5)
> plate6<-well.pos(6)
> plate7<-well.pos(7)
> plate8<-well.pos(8)
> plate<-rbind(cbind(plate1,plate=1),cbind(plate2,plate=2),
+              cbind(plate3,plate=3),cbind(plate4,plate=4),cbind(plate5,plate=5),
+              cbind(plate6,plate=6),cbind(plate7,plate=7),cbind(plate8,plate=8))
> pc1.all.col<-levelplot(pc1~num*let | factor(plate),as.table=TRUE,
+                        data=plate,col.regions=topo.colors(100),xlab="",ylab="",
+                        strip=strip.custom(bg="white"))
> pc2.all.col<-levelplot(pc2~num*let | factor(plate),as.table=TRUE,
+                        data=plate,col.regions=topo.colors(100),xlab="",ylab="",
+                        strip=strip.custom(bg="white"))
> jpeg(file=paste(outplot,"GENEMAM_heatmapPC1_color.jpg",sep=""))
> plot(pc1.all.col)
> dev.off()
> jpeg(file=paste(outplot,"GENEMAM_heatmapPC2_color.jpg",sep=""))
> plot(pc2.all.col)
> dev.off()
The sensitivity to filtering analysis and the run time analysis were performed using syntax similar to that used for the simulated data previously.

The syntax used to perform the ANOVA filtering analysis using the \texttt{limma} package is as follows and reproduces Table 4.2.

\begin{verbatim}
> plate<-demo$plate
> batch<-ifelse(plate<5,1,ifelse(plate==5,2,3))
> dat<-t(x2.imp) ## t(x2.imp) is pxn
> design<-model.matrix(~as.factor(plate)-1)
> colnames(design)<-paste("plate",1:8,sep="")
> fit<-lmFit(object=dat,design=design)
> fit2<-eBayes(fit)
> ### Using Benjamini & Hochberg Adjustment:
> result<-topTable(fit2,number=dim(x2.imp)[2],
+   sort.by="none",adjust="BH")
> ## alpha = 0.05
> sum(result$adj.P.Val<0.05)
> res.05<-ifelse(result$adj.P.Val<0.05,1,0)
> length(res.05)
> dim(x2.imp)
> data.anova.05<-x2.imp[,res.05==1]
> ## alpha = 0.01
> sum(result$adj.P.Val<0.01)
> res.01<-ifelse(result$adj.P.Val<0.01,1,0)
> length(res.01)
> data.anova.01<-x2.imp[,res.01==1]
> ## Using Bonferroni Adjustment:
> result2<-topTable(fit2,number=dim(x2.imp)[2],
+   sort.by="none",adjust="bonferroni")
> ## alpha = 0.05
> sum(result2$adj.P.Val<0.05)
> res.bon<-ifelse(result2$adj.P.Val<0.05,1,0)
> length(res.bon)
> data.anova.bon<-x2.imp[,res.bon==1]
> ## alpha = 0.01
> sum(result2$adj.P.Val<0.01)
\end{verbatim}
> res.bon.01<-ifelse(result2$adj.P.Val<0.01,1,0)
> length(res.bon.01)
> data.anova.bon.01<-x2.imp[,res.bon.01==1]
> ## Output Table
> adjust.method<-c(rep("BH",2),rep("Bon",2))
> alpha.lvl<-rep(c(0.05,0.01),2)
> retain<-c(sum(res.05),sum(res.01),sum(res.bon),sum(res.bon.01))
> tab<-data.frame(adjust.method,alpha.lvl,retain)

A.1.4 Chapter 5: Comparison of Batch Effect Adjustment Methods

In Chapter 5 we applied various batch effect correction methods to simulated and case study data. The data were simulated using the following syntax and these data were used throughout this chapter for all further analyses.

> n=90 ; p=1000
> grid<-expand.grid(n,p)
> names(grid)<-c('n','p')
> dim(grid)
> grid
> ## No Phenotype
> system.time(
+ simDat<-sim.data(grid=grid,g=1,b=2,s=0.5,bprop=0.01,beffect=c(0,1))
+ )
> save(simDat,file=paste(datafile,"NoPheData.RData",sep=""))
> ## High Variance Phenotype
> system.time(
+ simDat<-sim.data(grid=grid,g=1,b=2,s=0.5,bprop=0.03,beffect=c(0,1),
+ nphe=2,sp=2,pprop=0.1,peffect=c(0,1))
+ )
> save(simDat,file=paste(datafile,"HighPheData.RData",sep=""))
> ## Low Variance Phenotype
> system.time(
+ simDat<-sim.data(grid=grid,g=1,b=2,s=0.5,bprop=0.05,beffect=c(0,1),
+ nphe=2,sp=0.2,pprop=0.1,peffect=c(0,1))
+ )
> save(simDat,file=paste(datafile,"LowPheData.RData",sep=""))
## True Phenotype with Dependent Batch and Phenotype

```r
system.time(
+ simDat<-NewCoPheSim(n=90,p=1000,b=2,nphe=2,s=0.5,pprop=0.1,
+ beta_b=2,beta_p=0.5,set=TRUE,
+ nphegenes=50,nbatgenes=50,nphebatgenes=100)
+ )
> save(simDat,file=paste(datafile,"NewCoPheData_set.RData",sep=""))
```

The batch correction methods we applied to the data sets were BMC using the `pamr.batchadjust()` function in the `pamr` package, DWD using the `kdwd()` function in the `DWD` package, and empirical Bayes using the `ComBat()` function in the `sva` package. Examples of calls to these functions using the low variance phenotype data and including calls to `gPCA.batchdetect()` before and after batch correction by each method are

```r
load(file=paste(datafile,"LowPheData.RData",sep=""))
rawdata<-simDat$x
batch<-simDat$batch
pheno<-simDat$phes
## Apply gPCA to raw data
out.raw<-gPCA.batchdetect(x=rawdata,batch=batch,center=FALSE)
out.raw$delta ; out.raw$p.val
## Batch Correction using BMC
> time.pamr<-system.time(
+ pamrout<-pamr.batchadjust(data=list(x=t(rawdata),batchlabels=batch))
+ )[[3]]
> pamradj<-t(pamrout$x)
> out.adj.pamr<-gPCA.batchdetect(x=pamradj,batch=batch,center=FALSE)
> out.adj.pamr$delta ; out.adj.pamr$p.val
> ## Batch Correction using mDWD
> time.dwd<-system.time(
+ dwdout<-kdwd(x=as.factor(batch)~.,data=rawdata,scaled=FALSE,type="mdwd")
+ )[[3]]
> mdwd.adjust<-mdwd.batchadjust(dwdout=dwdout,batch=batch,
+ rawdata=as.matrix(rawdata))
> mdwdadj<-mdwd.adjust$data
> ## Apply gPCA to batch corrected data
```
> out.adj.dwd<-gPCA.batchdetect(x=mdwdadj,batch=batch,center=FALSE)
> out.adj.dwd$delta ; out.adj.dwd$p.val
> # Batch Correction using Non-parametric ComBat
> time.ebn<-system.time(
+ ebout_n<-ComBat(dat=t(rawdata),batch=batch,mod=model.matrix(~pheno),
+ numCovs=2,par.prior=FALSE)
+ )[[3]]
> ebnadj<-t(ebout_n)
> ## Apply gPCA to batch corrected data
> out.adj.ebn<-gPCA.batchdetect(x=ebnadj,batch=batch,center=FALSE)
> out.adj.ebn$delta ; out.adj.ebn$p.val
> # Batch Correction using Parametric ComBat
> time.ebp<-system.time(
+ ebout_p<-ComBat(dat=t(rawdata),batch=batch,mod=model.matrix(~pheno),
+ numCovs=2,par.prior=TRUE)
+ )[[3]]
> ebpadj<-t(ebout_p)
> ## Apply gPCA to batch corrected data
> out.adj.ebp<-gPCA.batchdetect(x=ebpadj,batch=batch,center=FALSE)
> out.adj.ebp$delta ; out.adj.ebp$p.val
> deltas<-c(out.raw$delta,out.adj.pamr$delta,out.adj.dwd$delta,
+ out.adj.ebn$delta,out.adj.ebp$delta)
> pvals<-c(out.raw$p.val,out.adj.pamr$p.val,out.adj.dwd$p.val,
+ out.adj.ebn$p.val,out.adj.ebp$p.val)
> times<-c(NA,time.pamr,time.dwd,time.ebn,time.ebp)
> lowphe<-data.frame(deltas,pvals,times)

Similar code is used for all simulated and case study data. The mdwd.batchadjust() function was created to perform the actual batch adjustment from the output of the kdwd() function.

> mdwd.batchadjust

function (dwdout, batch, rawdata)
{
  b <- length(unique(batch))
  if (class(rawdata) != "matrix") {

stop("Error: rawdata must have class 'matrix'.")
}
dirmx <- dwdout@w
vproj <- meanproj <- adj <- list()
adjdata <- matrix(nrow = dim(rawdata)[1], ncol = dim(rawdata)[2])
for (k in 1:b) {
  vproj[[k]] <- rawdata[batch == k, ] %*% dirmx
  meanproj[[k]] <- colMeans(vproj[[k]])
  adj[[k]] <- matrix(rep(dirmx %*% meanproj[[k]], dim(vproj[[k]])[1]),
                    nrow = dim(vproj[[k]])[1], byrow = TRUE)
  adjdata[batch == k, ] <- rawdata[batch == k] - adj[[k]]
}
list(data = adjdata)
}

To perform the batch correction analysis in Chapter 5, the limma package was used as follows for the high variance phenotype simulated data. This code reproduces the contingency tables in Figure 5.3

```r
> load(file=paste(datafile,"HighPheData.RData",sep=""))
> rawdata<-simDat$x
> batch<-simDat$batch
> pheno<-simDat$pheno
> ## Pre-correction gPCA
> out.pre<-gPCA.batchdetect(x=as.matrix(rawdata),batch=batch,center=FALSE)
> out.pre$delta ; out.pre$p.val
> # Batch Correction using BMC
> time.pamr<-system.time(
+ pamrout<-pamr.batchadjust(data=list(x=t(rawdata),batchlabels=batch))
+ )[[3]]
> pamradj<-t(pamrout$x)
> out.adj.pamr<-gPCA.batchdetect(x=pamradj,batch=batch,center=FALSE)
> out.adj.pamr$delta ; out.adj.pamr$p.val
> ## lmFit and eBayes to find significant features pre- and post-correction
> ## Pre-Batch Correction
>
> data<-t(simDat$x2)
```
true.phe <- c(rep(TRUE, simDat$p.prop*simDat$p),
              rep(FALSE, simDat$p - simDat$p.prop*simDat$p))
design <- model.matrix(~as.factor(pheno) - 1)
colnames(design) <- paste("pheno", 1:simDat$nphe, sep = "")
fit <- lmFit(data, design)
names(fit)
contr.matrix <- makeContrasts(compare = pheno1 - pheno2, levels = design)
fit2 <- contrasts.fit(fit, contr.matrix)
fit3 <- eBayes(fit2)
result <- topTable(fit3, number = dim(data)[1], sort.by = "none",
                   adjust = "BH")
sum(result$adj.P.Val < 0.10)
which(result$adj.P.Val < 0.10)
sig.feat.raw <- ifelse(result$adj.P.Val < 0.1, TRUE, FALSE)
res <- ifelse(result$adj.P.Val < 0.10, 1, 0)
## Post-Batch Correction: BMC
fit_pamr <- lmFit(pamrout$x, design)
fit2_pamr <- contrasts.fit(fit_pamr, contr.matrix)
fit3_pamr <- eBayes(fit2_pamr)
result_pamr <- topTable(fit3_pamr, number = dim(pamrout$x)[1],
                        sort.by = "none", adjust = "BH")
sum(result_pamr$adj.P.Val < 0.10)
which(result_pamr$adj.P.Val < 0.10)
sig.feat.pamr <- ifelse(result_pamr$adj.P.Val < 0.1, TRUE, FALSE)
res_pamr <- ifelse(result_pamr$adj.P.Val < 0.10, 1, 0)
resmx_pamr <- cbind(res, res_pamr)
# png(file = paste(outplot, "PAMR_Venn_DiPwr_p.png", sep = ""))
# vennDiagram(resmx_pamr, names = c("Raw Data", "Corrected Data"))
# dev.off()
table(sig.feat.raw, sig.feat.pamr)
table(true.phe, sig.feat.raw)
table(true.phe, sig.feat.pamr)
## Batch Correction Using Empirical Bayes
mod <- model.matrix(~as.factor(pheno))
dim(mod)
names(mod) <- paste("pheno", 1:simDat$nphe)
time.ebn <- system.time(
  diComBat_n <- ComBat(dat = data, batch = batch, mod = mod, par.prior = FALSE)
+ )[[3]]
> time ebp <- system.time(
+ diComBat_p <- ComBat(dat=data, batch=batch, mod=mod, par.prior=TRUE)
+ )[[3]]
> out adj ebn <- gPCA batchdetect(x=t(diComBat_n), batch=batch, center=FALSE)
> out adj ebn$delta ; out adj ebn$p.val
> out adj ebp <- gPCA batchdetect(x=t(diComBat_p), batch=batch, center=FALSE)
> out adj ebp$delta ; out adj ebp$p.val
> ## Post-Batch Correction: Non-parametric EB
> fit n <- lmFit(diComBat_n, design)
> contr matrix <- makeContrasts(compare=pheno1-pheno2, levels=design)
> fit2 n <- contrasts.fit(fit n, contr matrix)
> fit3 n <- eBayes(fit2 n)
> result n <- topTable(fit3 n, number=dim(diComBat_n)[1], sort.by="none",
+ adjust="BH")
> sum(result n$adj.P.Val<0.10)
> which(result n$adj.P.Val<0.10)
> sig feat n <- ifelse(result n$adj.P.Val<0.1, TRUE, FALSE)
> res n <- ifelse(result n$adj.P.Val<0.10, 1, 0)
> resmx n <- cbind(res, res n)
> # png(file=paste(outplot, "ComBat_Venn_DiPwr_n.png", sep=""))
> # vennDiagram(resmx n, names=c("Raw Data", "EB Corrected Data"))
> # dev.off()
> table(sig feat raw, sig feat n)
> table(true phe, sig feat raw)
> table(true phe, sig feat n)
> ## Post-Batch Correction: Parametric EB
> fit p <- lmFit(diComBat_p, design)
> fit2 p <- contrasts.fit(fit p, contr matrix)
> fit3 p <- eBayes(fit2 p)
> result p <- topTable(fit3 p, number=dim(diComBat_p)[1], sort.by="none",
+ adjust="BH")
> sum(result p$adj.P.Val<0.10)
> which(result p$adj.P.Val<0.10)
> sig feat p <- ifelse(result p$adj.P.Val<0.1, TRUE, FALSE)
> res p <- ifelse(result p$adj.P.Val<0.10, 1, 0)
> resmx p <- cbind(res, res p)
> # png(file=paste(outplot, "ComBat_Venn_HighPwr_p.png", sep=""))
A.1.5 Chapter 6: The gPCA Package for Identifying Batch Effects

All code used in this chapter can be found in the gPCA package. The gPCA package consists of the gPCA::batchdetect() function provided previously, three visualization functions that plot the data in different ways, and four example data sets. Chapter 6 describes the functions and syntax.
Vita

Sarah Elizabeth Reese was born on January 11, 1985 in Washington, D.C., and is an American citizen. She graduated from Connelly School of the Holy Child in Potomac, MD in 2003. She received her Bachelor of Science in Mathematics from the University of Mary Washington in Fredericksburg, VA in 2007.