Correcting batch effects in single-cell RNA-seq data

***Aaron T. L. Lun*[1] *and Michael D. Morgan*[2]**

[1]Cancer Research UK Cambridge Institute, Li Ka Shing Centre, Robinson Way, Cambridge CB2 0RE, United Kingdom
[2]Wellcome Trust Sanger Institute, Wellcome Genome Campus, Hinxton, Cambridge CB10 1SA, United Kingdom

2019-01-09

1 Introduction

Large single-cell RNA sequencing (scRNA-seq) projects usually need to generate data across multiple batches due to logistical constraints. However, the processing of different batches is often subject to uncontrollable differences, e.g., changes in operator, differences in reagent quality. This results in systematic differences in the observed expression in cells from different batches, which we refer to as "batch effects". Batch effects are problematic as they can be major drivers of heterogeneity in the data, masking the relevant biological differences and complicating interpretation of the results.

Computational correction of these effects is critical for eliminating batch-to-batch variation, allowing data across multiple batches to be combined for valid downstream analysis. However, existing methods such as `removeBatchEffect()` (Ritchie et al. 2015) assume that the composition of cell populations are either known or the same across batches. This workflow describes the application of an alternative strategy for batch correction based on the detection of mutual nearest neighbours (MNNs) (Haghverdi et al. 2018). The MNN approach does not rely on pre-defined or equal population compositions across batches, only requiring that a subset of the population be shared between batches. We demonstrate its use on two human pancreas scRNA-seq datasets generated in separate studies.

2 Processing the different datasets

2.1 CEL-seq, GSE81076

2.1.1 Loading in the data

This dataset was generated by Grun et al. (2016) using the CEL-seq protocol with unique molecular identifiers (UMIs) and ERCC spike-ins. Count tables were obtained from the NCBI Gene Expression Omnibus using the accession number above.

```
library(BiocFileCache)
bfc <- BiocFileCache("raw_data", ask = FALSE)
grun.fname <- bfcrpath(bfc, file.path("ftp://ftp.ncbi.nlm.ni
h.gov/geo/series",
    "GSE81nnn/GSE81076/suppl/GSE81076%5FD2%5F3%5F7%5F10%5F1
7%2Etxt%2Egz"))
```

We first read the table into memory.

```
gse81076.df <- read.table(grun.fname, sep='\t',
    header=TRUE, stringsAsFactors=FALSE, row.names=1)
dim(gse81076.df)
```

```
## [1] 20148  1728
```

Unfortunately, the data and metadata are all mixed together in this file. As a result, we need to manually extract the metadata from the column names.

```
donor.names <- sub("^(D[0-9]+).*", "\\1", colnames(gse81076.
df))
table(donor.names)
```

```
## donor.names
##    D101    D102  D10631     D17   D1713 D172444      D2
D3     D71
##      96      96      96     288      96      96      96
480      96
##     D72     D73     D74
##      96      96      96
```

```
plate.id <- sub("^D[0-9]+(.*)_.*", "\\1", colnames(gse81076.
df))
table(plate.id)
```

```
## plate.id
##      All1 All2 TGFB  en1  en2  en3  en4   ex
##  864   96   96   96   96   96   96   96  192
```

Another irritating feature of this dataset is that gene symbols were supplied, rather than stable identifiers such as Ensembl. We convert all row names to Ensembl identifiers, removing `NA` or duplicated entries (with the exception of spike-in transcripts).

```
gene.symb <- gsub("__chr.*$", "", rownames(gse81076.df))
is.spike <- grepl("^ERCC-", gene.symb)
table(is.spike)
```

```
## is.spike
## FALSE  TRUE
## 20064    84
```

```
library(org.Hs.eg.db)
gene.ids <- mapIds(org.Hs.eg.db, keys=gene.symb, keytype="SY
MBOL", column="ENSEMBL")
gene.ids[is.spike] <- gene.symb[is.spike]

keep <- !is.na(gene.ids) & !duplicated(gene.ids)
gse81076.df <- gse81076.df[keep,]
rownames(gse81076.df) <- gene.ids[keep]
summary(keep)
```

```
##    Mode   FALSE    TRUE
## logical    2071   18077
```

We create a `SingleCellExperiment` object to store the counts and metadata together. This reduces the risk of book-keeping errors in later steps of the analysis. Note that we re-identify the spike-in rows, as the previous indices would have changed after the subsetting.

```
library(SingleCellExperiment)
sce.gse81076 <- SingleCellExperiment(list(counts=as.matrix(g
se81076.df)),
    colData=DataFrame(Donor=donor.names, Plate=plate.id),
    rowData=DataFrame(Symbol=gene.symb[keep]))
isSpike(sce.gse81076, "ERCC") <- grepl("^ERCC-", rownames(gs
e81076.df))
sce.gse81076
```

```
## class: SingleCellExperiment
## dim: 18077 1728
## metadata(0):
## assays(1): counts
## rownames(18077): ENSG00000268895 ENSG00000121410 ... ENSG
00000074755
##   ENSG00000036549
## rowData names(1): Symbol
## colnames(1728): D2ex_1 D2ex_2 ... D17TGFB_95 D17TGFB_96
## colData names(2): Donor Plate
## reducedDimNames(0):
## spikeNames(1): ERCC
```

2.1.2 Quality control and normalization

We compute quality control (QC) metrics for each cell (McCarthy et al. 2017) and identify cells with low library sizes, low numbers of expressed genes, or high ERCC content.

```
library(scater)
sce.gse81076 <- calculateQCMetrics(sce.gse81076, compact=TRU
E)
QC <- sce.gse81076$scater_qc
low.lib <- isOutlier(QC$all$log10_total_counts, type="lowe
r", nmad=3)
low.genes <- isOutlier(QC$all$log10_total_features_by_count
s, type="lower", nmad=3)
high.spike <- isOutlier(QC$feature_control_ERCC$pct_counts,
type="higher", nmad=3)
data.frame(LowLib=sum(low.lib), LowNgenes=sum(low.genes),
    HighSpike=sum(high.spike, na.rm=TRUE))
```

```
##   LowLib LowNgenes HighSpike
## 1     55       130       388
```

Cells with extreme values for these QC metrics are presumed to be of low quality and are removed. A more thorough analysis would examine the distributions of these QC metrics beforehand, but we will skip that step for brevity here.

```
discard <- low.lib | low.genes | high.spike
sce.gse81076 <- sce.gse81076[,!discard]
summary(discard)
```

```
##    Mode   FALSE    TRUE
## logical    1292     436
```

We compute size factors for the endogenous genes using the deconvolution method (Lun, Bach, and Marioni 2016). This is done with pre-clustering through `quickCluster()` to avoid pooling together very different cells.

```
library(scran)
set.seed(1000)
clusters <- quickCluster(sce.gse81076, method="igraph", min.
mean=0.1)
table(clusters)
```

```
## clusters
##   1   2   3
## 513 331 448
```

```
sce.gse81076 <- computeSumFactors(sce.gse81076, min.mean=0.
1, clusters=clusters)
summary(sizeFactors(sce.gse81076))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.002648 0.442511 0.797307 1.000000 1.296612 9.507820
```

We also compute size factors for the spike-in transcripts (Lun et al. 2017). Recall that we set `general.use=FALSE` to ensure that the spike-in size factors are only applied to the spike-in transcripts.

```
sce.gse81076 <- computeSpikeFactors(sce.gse81076, general.us
e=FALSE)
summary(sizeFactors(sce.gse81076, "ERCC"))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01042 0.57782 0.88699 1.00000 1.27765 7.43998
```

We then compute normalized log-expression values for use in downstream analyses.

```
sce.gse81076 <- normalize(sce.gse81076)
```

2.1.3 Identifying highly variable genes

We identify highly variable genes (HVGs) using `trendVar()` and `decomposeVar()`, using the variances of spike-in transcripts to model technical noise. We set `block=` to ensure that uninteresting differences between plates or donors do not inflate the variance. The small discrepancy in the fitted trend in Figure 1 is caused by the fact that the trend is fitted robustly to the block-wise variances of the spike-ins, while the variances shown are averaged across blocks and not robust to outliers.

```
block <- paste0(sce.gse81076$Plate, "_", sce.gse81076$Donor)
fit <- trendVar(sce.gse81076, block=block, parametric=TRUE)
dec <- decomposeVar(sce.gse81076, fit)

plot(dec$mean, dec$total, xlab="Mean log-expression",
    ylab="Variance of log-expression", pch=16)
is.spike <- isSpike(sce.gse81076)
points(dec$mean[is.spike], dec$total[is.spike], col="red", p
ch=16)
curve(fit$trend(x), col="dodgerblue", add=TRUE)
```

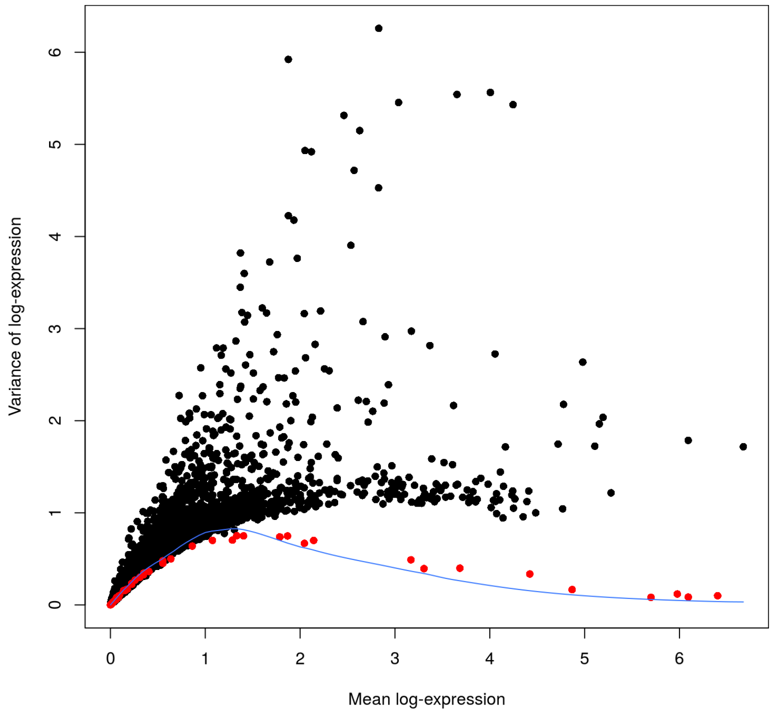


Figure 1: Variance of normalized log-expression values for each gene in the GSE81076 dataset, plotted against the mean log-expression. The blue line represents the mean-dependent trend fitted to the variances of the spike-in transcripts (red).

We order genes by decreasing biological component, revealing some usual suspects such as insulin and glucagon. We will be using this information later when performing feature selection prior to running `mnnCorrect()` .

```
dec.gse81076 <- dec
dec.gse81076$Symbol <- rowData(sce.gse81076)$Symbol
dec.gse81076 <- dec.gse81076[order(dec.gse81076$bio, decreas
ing=TRUE),]
head(dec.gse81076)
```

```
## DataFrame with 6 rows and 7 columns
##                              mean            total
bio
##                         <numeric>        <numeric>
<numeric>
## ENSG00000254647 2.82926326785746 6.25972220963598 5.81569
316873789
## ENSG00000129965 1.87615109970638 5.92173614351272 5.47357
562769792
## ENSG00000115263 4.00610832326079 5.56338526078797 5.33005
501610815
## ENSG00000118271 3.65393642101131 5.54285843054711 5.24069
541798718
## ENSG00000115386 4.24556465621839 5.43121995552112 5.19578
897832117
## ENSG00000164266 3.03848794345203  5.4545366051026 5.04804
616712653
##                              tech   p.value       FDR
Symbol
##                         <numeric> <numeric> <numeric> <ch
aracter>
## ENSG00000254647 0.444029040898086         0         0
INS
## ENSG00000129965 0.448160515814804         0         0
INS-IGF2
## ENSG00000115263 0.233330244679815         0         0
GCG
## ENSG00000118271 0.302163012559924         0         0
TTR
## ENSG00000115386 0.235430977199958         0         0
REG1A
## ENSG00000164266 0.406490437976075         0         0
SPINK1
```

2.2 CEL-seq2, GSE85241

2.2.1 Loading in the data

This dataset was generated by Muraro et al. (2016) using the CEL-seq2 protocol with unique molecular identifiers (UMIs) and ERCC spike-ins. Count tables were obtained from the NCBI Gene Expression Omnibus using the accession number above.

```
muraro.fname <- bfcrpath(bfc, file.path("ftp://ftp.ncbi.nlm.
nih.gov/geo/series",
    "GSE85nnn/GSE85241/suppl",
    "GSE85241%5Fcellsystems%5Fdataset%5F4donors%5Fupdated%2E
csv%2Egz"))
```

We first read the table into memory.

```
gse85241.df <- read.table(muraro.fname, sep='\t',
    header=TRUE, row.names=1, stringsAsFactors=FALSE)
dim(gse85241.df)
```

```
## [1] 19140  3072
```

We extract the metadata from the column names.

```
donor.names <- sub("^(D[0-9]+).*", "\\1", colnames(gse85241.
df))
table(donor.names)
```

```
## donor.names
## D28 D29 D30 D31
## 768 768 768 768
```

```
plate.id <- sub("^D[0-9]+\\.([0-9]+)_.*", "\\1", colnames(gs
e85241.df))
table(plate.id)
```

```
## plate.id
##   1   2   3   4   5   6   7   8
## 384 384 384 384 384 384 384 384
```

Yet again, gene symbols were supplied instead of Ensembl or Entrez identifiers. We convert all row names to Ensembl identifiers, removing `NA` or duplicated entries (with the exception of spike-in transcripts).

```
gene.symb <- gsub("__chr.*$", "", rownames(gse85241.df))
is.spike <- grepl("^ERCC-", gene.symb)
table(is.spike)
```

```
## is.spike
## FALSE  TRUE
## 19059    81
```

```
library(org.Hs.eg.db)
gene.ids <- mapIds(org.Hs.eg.db, keys=gene.symb, keytype="SY
MBOL", column="ENSEMBL")
gene.ids[is.spike] <- gene.symb[is.spike]

keep <- !is.na(gene.ids) & !duplicated(gene.ids)
gse85241.df <- gse85241.df[keep,]
rownames(gse85241.df) <- gene.ids[keep]
summary(keep)
```

```
##    Mode   FALSE    TRUE
## logical    1949   17191
```

We create a `SingleCellExperiment` object to store the counts and metadata together.

```
sce.gse85241 <- SingleCellExperiment(list(counts=as.matrix(g
se85241.df)),
    colData=DataFrame(Donor=donor.names, Plate=plate.id),
    rowData=DataFrame(Symbol=gene.symb[keep]))
isSpike(sce.gse85241, "ERCC") <- grepl("^ERCC-", rownames(gs
e85241.df))
sce.gse85241
```

```
## class: SingleCellExperiment
## dim: 17191 3072
## metadata(0):
## assays(1): counts
## rownames(17191): ENSG00000268895 ENSG00000121410 ... ENSG
00000074755
##   ENSG00000036549
## rowData names(1): Symbol
## colnames(3072): D28.1_1 D28.1_2 ... D30.8_95 D30.8_96
## colData names(2): Donor Plate
## reducedDimNames(0):
## spikeNames(1): ERCC
```

2.2.2 Quality control and normalization

We compute QC metrics for each cell and identify cells with low library sizes, low numbers of expressed genes, or high ERCC content.

```
sce.gse85241 <- calculateQCMetrics(sce.gse85241, compact=TRU
E)
QC <- sce.gse85241$scater_qc
low.lib <- isOutlier(QC$all$log10_total_counts, type="lowe
r", nmad=3)
low.genes <- isOutlier(QC$all$log10_total_features_by_count
s, type="lower", nmad=3)
high.spike <- isOutlier(QC$feature_control_ERCC$pct_counts,
type="higher", nmad=3)
data.frame(LowLib=sum(low.lib), LowNgenes=sum(low.genes),
    HighSpike=sum(high.spike, na.rm=TRUE))
```

```
##   LowLib LowNgenes HighSpike
## 1    577       669       696
```

Low-quality cells are defined as those with extreme values for these QC metrics and are removed.

```
discard <- low.lib | low.genes | high.spike
sce.gse85241 <- sce.gse85241[,!discard]
summary(discard)
```

```
##    Mode   FALSE    TRUE
## logical    2346     726
```

We compute size factors for the endogenous genes and spike-in transcripts, and use them to compute log-normalized expression values.

```
set.seed(1000)
clusters <- quickCluster(sce.gse85241, min.mean=0.1, method
="igraph")
table(clusters)
```

```
## clusters
##   1   2   3   4   5   6
## 237 248 285 483 613 480
```

```
sce.gse85241 <- computeSumFactors(sce.gse85241, min.mean=0.
1, clusters=clusters)
summary(sizeFactors(sce.gse85241))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##  0.07856  0.53931  0.81986  1.00000  1.22044 14.33280
```

```
sce.gse85241 <- computeSpikeFactors(sce.gse85241, general.us
e=FALSE)
summary(sizeFactors(sce.gse85241, "ERCC"))
```

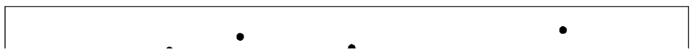
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.09295 0.61309 0.88902 1.00000 1.27519 4.04643
```

```
sce.gse85241 <- normalize(sce.gse85241)
```

2.2.3 Identifying highly variable genes

We fit a trend to the spike-in variances as previously described, allowing us to model the technical noise for each gene (Figure 2). Again, we set `block=` to ensure that uninteresting differences between plates or donors do not inflate the variance.

```
block <- paste0(sce.gse85241$Plate, "_", sce.gse85241$Donor)
fit <- trendVar(sce.gse85241, block=block, parametric=TRUE)
dec <- decomposeVar(sce.gse85241, fit)
plot(dec$mean, dec$total, xlab="Mean log-expression",
    ylab="Variance of log-expression", pch=16)
is.spike <- isSpike(sce.gse85241)
points(dec$mean[is.spike], dec$total[is.spike], col="red", p
ch=16)
curve(fit$trend(x), col="dodgerblue", add=TRUE)
```



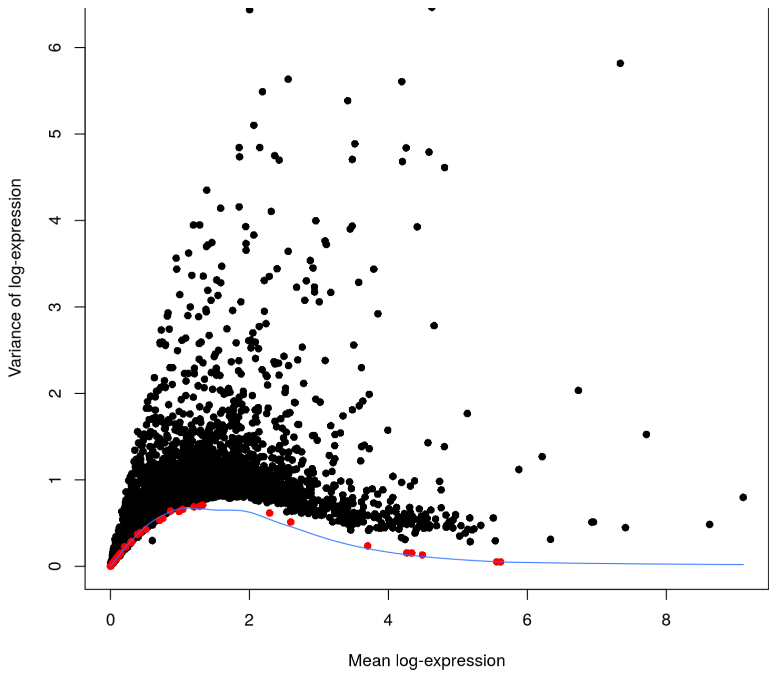


Figure 2: Variance of normalized log-expression values for each gene in the GSE85241 dataset, plotted against the mean log-expression. The blue line represents the mean-dependent trend fitted to the variances of the spike-in transcripts (red).

We order genes by decreasing biological component, as described above.

```
dec.gse85241 <- dec
dec.gse85241$Symbol <- rowData(sce.gse85241)$Symbol
dec.gse85241 <- dec.gse85241[order(dec.gse85241$bio, decreas
ing=TRUE),]
head(dec.gse85241)
```

```
## DataFrame with 6 rows and 7 columns
##                              mean            total
bio
##                         <numeric>        <numeric>
<numeric>
## ENSG00000115263 7.66453729345785 6.66863456231166 6.63983
282674494
## ENSG00000089199 4.62375793902937 6.46558866721711 6.34422
879524315
## ENSG00000169903 3.01813483888172 6.59298310984116 6.23983
239174791
## ENSG00000254647 2.00308741950834 6.43736230365307 5.88133
815055707
## ENSG00000118271 7.33751241351268 5.81792529163505  5.7864
800378735
## ENSG00000171951  4.1933909879622 5.60615948812391 5.44209
845614432
##                                tech   p.value       FDR
Symbol
##                           <numeric> <numeric> <numeric> <c
haracter>
## ENSG00000115263 0.0288017355667183         0         0
GCG
## ENSG00000089199  0.121359871973959         0         0
CHGB
## ENSG00000169903  0.353150718093246         0         0
TM4SF4
## ENSG00000254647  0.556024153095996         0         0
INS
## ENSG00000118271 0.0314452537615524         0         0
TTR
## ENSG00000171951  0.164061031979585         0         0
SCG2
```

2.3 Comments on additional batches

In Haghverdi et al. (2018), we originally performed batch correction across four separate pancreas scRNA-seq datasets. For simplicity, we will only consider the two batches generated using CEL-seq(2) and ignore those generated using Smart-seq2 (Segerstolpe et al. 2016; Lawlor et al. 2017). As one might expect, batch correction is easiest when dealing with data generated from the same technology, as fewer systematic differences are present that can interfere with the biological structure. Nonetheless, it is often possible to obtain good results when applying MNN correction to batches of data generated with different technologies.

It is also worth pointing out that both of the CEL-seq(2) batches above contain cells from multiple donors. Each donor could be treated as a separate batch in their own right, reflecting (presumably uninteresting) biological differences between donors due to genotype, age, sex or other factors that are not easily controlled when dealing with humans. For simplicity, we will ignore the donor effects within each study and only consider the removal of the batch effect between the two studies. However, we note that it is possible to apply the MNN correction between donors in each batch and then between the batches - see `?fastMNN` for details.

3 Feature selection across batches

To obtain a single set of features for batch correction, we compute the average biological component across all batches. We then take all genes with positive biological components to ensure that all interesting biology is retained, equivalent to the behaviour of `denoisePCA()`. However, the quality of the correction can often be sensitive to technical noise, which means that some discretion may be required during feature selection. Users may prefer to take the top 1000-5000 genes with the largest average components, or to use `combineVar()` to obtain combined *p*-values for gene selection.

```
universe <- intersect(rownames(dec.gse85241), rownames(dec.g
se81076))
mean.bio <- (dec.gse85241[universe,"bio"] + dec.gse81076[uni
verse,"bio"])/2
chosen <- universe[mean.bio > 0]
length(chosen)
```

```
## [1] 14758
```

We also rescale each batch to adjust for differences in sequencing depth between batches. The `multiBatchNorm()` function recomputes log-normalized expression values after adjusting the size factors for systematic differences in coverage between `SingleCellExperiment` objects. (Keep in mind that the previously computed size factors only remove biases between cells *within* a single batch.) This improves the quality of the correction by removing one aspect of the technical differences between batches.

```
rescaled <- multiBatchNorm(sce.gse85241[universe,], sce.gse8
1076[universe,])
rescaled.gse85241 <- rescaled[[1]]
rescaled.gse81076 <- rescaled[[2]]
```

Comments from Aaron:

- Technically, we should have performed variance modelling and feature selection *after* calling `multiBatchNorm()`. This ensures that the variance components are estimated from the same values to be used in the batch correction. In practice, this makes little difference, and it tends to be easier to process each batch separately and consolidate all results in one step as shown above.

4 Performing MNN-based correction

Consider a cell \(a\) in batch \(A\), and identify the cells in batch \(B\) that are nearest neighbours to \(a\) in the expression space defined by the selected features. Repeat this for a cell \(b\) in batch \(B\), identifying its nearest neighbours in \(A\). Mutual nearest neighbours are pairs of cells from different batches that belong in each other's set of nearest neighbours. The reasoning is that MNN pairs represent cells from the same biological state prior to the application of a batch effect - see Haghverdi et al. (2018) for full theoretical details. Thus, the difference between cells in MNN pairs can be used as an estimate of the batch effect, the subtraction of which can yield batch-corrected values.

We apply the `fastMNN()` function to the three batches to remove the batch effect, using the genes in `chosen`. To reduce computational work and technical noise, all cells in all cells are projected into the low-dimensional space defined by the top `d` principal components. Identification of MNNs and calculation of correction vectors are then performed in this low-dimensional space. The function returns a matrix of corrected values for downstream analyses like clustering or visualization.

```
set.seed(100)
original <- list(
    GSE81076=logcounts(rescaled.gse81076)[chosen,],
    GSE85241=logcounts(rescaled.gse85241)[chosen,]
)

# Slightly convoluted call to avoid re-writing code later.
# Equivalent to fastMNN(GSE81076, GSE85241, k=20, d=50, appr
oximate=TRUE)
mnn.out <- do.call(fastMNN, c(original, list(k=20, d=50, app
roximate=TRUE)))
dim(mnn.out$corrected)
```

```
## [1] 3638   50
```

Each row of the `corrected` matrix corresponds to a cell in one of the batches. The `batch` field contains a run-length encoding object specifying the batch of origin of each row.

```
mnn.out$batch
```

```
## character-Rle of length 3638 with 2 runs
##   Lengths:       1292       2346
##   Values : "GSE81076" "GSE85241"
```

Advanced users may also be interested in the list of `DataFrame` s in the `pairs` field. Each `DataFrame` describes the MNN pairs identified upon merging of each successive batch. This may be useful for checking the identified MNN pairs against known cell type identity, e.g., to determine if the cell types are being paired correctly.

```
mnn.out$pairs
```

```
## [[1]]
## DataFrame with 6635 rows and 2 columns
##          first    second
##      <integer> <integer>
## 1            1      1794
## 2            1      2087
## 3           15      1612
## 4           15      2512
## 5           15      2575
## ...        ...       ...
## 6631      1290      2339
## 6632      1290      2625
## 6633      1290      1538
## 6634      1290      2794
## 6635      1290      2001
```

As previously mentioned, we have only used two batches here to simplify the workflow. However, the MNN approach is not limited to two batches, and inclusion of more batches is as simple as adding more `SingleCellExperiment` objects to the `fastMNN()` call.

Comments from Aaron:

- The `k=` parameter specifies the number of nearest neighbours to consider when defining MNN pairs. This should be interpreted as the minimum frequency of each cell type or state in each batch. Larger values will improve the precision of the correction by increasing the number of MNN pairs, at the cost of reducing accuracy by allowing MNN pairs to form between cells of different type.
- The order of the supplied batches does matter, as all batches are corrected towards the first. We suggest setting the largest and/or most heterogeneous batch as the first. This ensures that sufficient MNN pairs will be identified between the first and other batches for stable correction. In situations where the nature of each batch is unknown, users can set `auto.order=TRUE` to allow `fastMNN()` to empirically

choose which batches to merge at each step. Batches are chosen to maximize the number of MNN pairs in order to provide a stable correction.

- When `approximate=TRUE`, `fastMNN()` uses methods from the *irlba (https://CRAN.R-project.org/package=irlba)* package to perform the principal components analysis quickly. While the run-to-run differences should be minimal, it does mean that `set.seed()` is required to obtain fully reproducible results.

5 Examining the effect of correction

We create a new `SingleCellExperiment` object containing log-expression values for all cells, along with information regarding the batch of origin. The MNN-corrected values are stored as dimensionality reduction results, befitting the principal components analysis performed within `fastMNN()`.

```
omat <- do.call(cbind, original)
sce <- SingleCellExperiment(list(logcounts=omat))
reducedDim(sce, "MNN") <- mnn.out$corrected
sce$Batch <- as.character(mnn.out$batch)
sce
```

```
## class: SingleCellExperiment
## dim: 14758 3638
## metadata(0):
## assays(1): logcounts
## rownames(14758): ENSG00000115263 ENSG00000089199 ... ENSG
00000148688
##   ENSG00000176731
## rowData names(0):
## colnames(3638): D2ex_1 D2ex_2 ... D30.8_93 D30.8_94
## colData names(1): Batch
## reducedDimNames(1): MNN
## spikeNames(0):
```

We examine the batch correction with some *t*-SNE plots. Figure~3 demonstrates how the cells separate by batch of origin in the uncorrected data. After correction, more intermingling between batches is observed, consistent with the removal of batch effects. Note that the E-MTAB-5601 dataset still displays some separation, which is probably due to the fact that the other batches are UMI datasets.

```
set.seed(100)
# Using irlba to set up the t-SNE, for speed.
osce <- runPCA(sce, ntop=Inf, method="irlba")
osce <- runTSNE(osce, use_dimred="PCA")
ot <- plotTSNE(osce, colour_by="Batch") + ggtitle("Origina
l")

set.seed(100)
csce <- runTSNE(sce, use_dimred="MNN")
ct <- plotTSNE(csce, colour_by="Batch") + ggtitle("Correcte
d")

multiplot(ot, ct, cols=2)
```

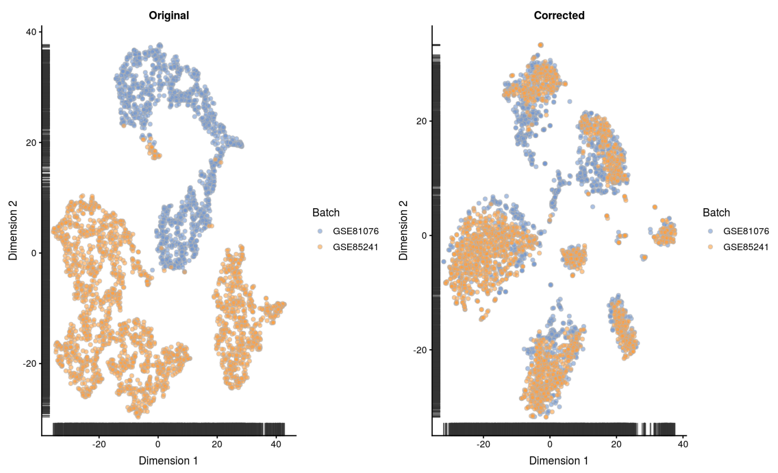


Figure 3: t-SNE plots of the pancreas datasets, before and after MNN correction. Each point represents a cell and is coloured by the batch of origin.

We colour by the expression of marker genes for known pancreas cell types to determine whether the correction is biologically sensible. Cells in the same visual cluster express the same marker genes (Figure 4), indicating that the correction maintains separation of cell types.

```
ct.gcg <- plotTSNE(csce, colour_by="ENSG00000115263") +
    ggtitle("Alpha cells (GCG)")
ct.ins <- plotTSNE(csce, colour_by="ENSG00000254647") +
    ggtitle("Beta cells (INS)")
ct.sst <- plotTSNE(csce, colour_by="ENSG00000157005") +
    ggtitle("Delta cells (SST)")
ct.ppy <- plotTSNE(csce, colour_by="ENSG00000108849") +
    ggtitle("PP cells (PPY)")
multiplot(ct.gcg, ct.ins, ct.sst, ct.ppy, cols=2)
```