Single-Cell Data Integration using MINT

Al J Abadi, Dr Kim-Anh Lê Cao

Melbourne Integrative Genomics
The University of Melbourne

2019-01-11

Contents

How to reproduce this vignette

You need the ‘bookdown’ package to reproduce this book. It is recommended to clone the repository, open the .Rproj file and load the Rmd files and create a ‘gitbook’ from the ‘build’ pane.

Alternatively, you can run the standalone vignettes from the *vignettes-standalone* folder.

```r
## install only if not installed
if (!requireNamespace('bookdown', quietly = TRUE)){
  paste('Trying to install Bookdown')
  install.packages('bookdown')
}
```

Chapter 1

Single Cell Data Integration Using MINT

This vignette explains the functionalities of **MINT** (**M**ultivariate **INT**egrative method) (Rohart et al., 2017) toolkit in combining datasets from multiple single cell transcriptomic studies on the same cell types. The integration is across the common P features. Hence, we call this framework **P-Integration**. We will also illustrate why Prinicipal Component Analysis might not be as powerful in such setting. It is worth mentioning that *MINT* can be used to combine datasets from any types of similar 'omics studies (transcriptomics, proteomics, metabolomics, etc) on the samples which share common features. For integration across different omics you can use DIABLO.

1.1 prerequisites

mixOmics (Le Cao et al., 2017) and the following Bioconductor/R packages must be installed for this vignette:

If any issues arise during package installations, you can check out the troubleshoots section.

1.2 why MINT?

Different datasets often contain different levels of unwanted variation that come from factors other than biology of study (e.g. differences in the technology, protocol, etc.) known as batch effects. The *MINT* framework presented in this vignette is an integrative method that helps to find the common structures in the combined dataset which best explain their biological class membership (cell type), and thus can potentially lead us to insightful signatures. For further examples on bulk data, you can also check out a case study on mixOmics' website.

1.3 when not to use MINT?

Current version of *MINT* performs supervised data integration. Therefore, the phenotypes/cell types must be known prior to data integration using this method.

1.4 R scripts

The R scripts are available at this link.

1.5 libraries

Here we load the libraries installed in prerequisites:

```r
## load the required libraries
library(SingleCellExperiment)
library(mixOmics)
library(scran)
library(knitr)
library(VennDiagram)
library(tibble)
```

1.6 data

Here we set up the input and output specified in *params* section of *yaml*. By default, all required data are loaded from GitHub and run data are not saved.

```r
check.exists <- function(object) ## function to assess existence of objects
{
  exists(as.character(substitute(object)))
}

## input/output from parameters
io = list()

## whether or where from to locally load data - FALSE: GitHub load; or a directory
io$local.input = ifelse(check.exists(params$local.input), params$local.input, F)

## whether or where to save run data - FALSE: do not save; or a directory
io$output.data = ifelse(check.exists(params$output.data), params$output.data, F)

## whether or where to save R scripts - FALSE: do not save; or a directory
io$Rscripts=ifelse(check.exists(params$Rscripts), params$Rscripts, F)
```

The benchmark data - available on GitHub - were obtained from our collaborators Luyi Tian and Dr Matt Ritchie at WEHI. Briefly, cells from three human cell lines H2228, H1975, HCC827 collected on lung tissue (Adenocarcinoma; Non-Small Cell Lung Cancer) were barcoded and pooled in equal amounts. The samples were then processed through three different types of 3' end sequencing protocols that span the range of isolation strategies available:

- Droplet-based capture with:
  - Chromium 10X (10X Genomics)
  - Drop-seq (Dolomite).
- Plate based isolation of cells in microwells with CEL-seq2.

Therefore, in addition to the biological variability coming from three distinct and separately cultured cell lines, the datasets are likely to also contain different layers of technical variability coming from three sequencing protocols and technologies.

Throughout this vignette, the terms *batch*, *protocol*, and *study* are used interchangeably.

1.7 loading the data

We can load the data either directly from the GitHub repository or from local directory into R environment:

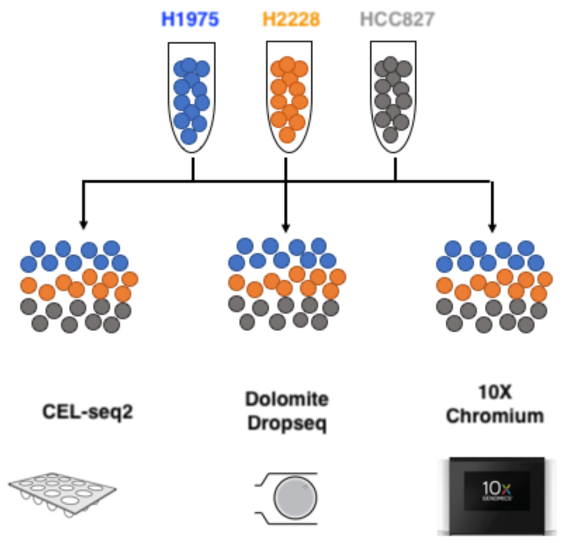


Figure 1.1: Benchmark experiment design: mixture of the three pure cells in equal proportion, processed through 3 different sequencing technologies. Adopted from Luyi Tian slides.

```r
if (isFALSE(io$local.input)){
  ## load from GitHub
  DataURL='https://tinyurl.com/sincell-with-class-RData-LuyiT'
  load(url(DataURL))
} else {
  stopifnot(dir.exists(io$local.input))
  load(file.path(io$local.input, 'sincell_with_class.RData'))
}
```

Tip: For convenience, throughout the runs you may also want to save the finalised RData file from each section using the *save* function for easy loading using *load* function in the downstream analyses (refer to R documentation for more details).

The loaded datasets consist of:

- *sce_sc_10x_qc*: From the Chromium 10X technology;
- *sce_sc_CELseq2_qc*: From the CEL-seq2 technology; and
- *sce_sc_Dropseq_qc*: From the Drop-seq technology (quality controlled twice due to abundant outliers).

Each dataset is of *SingleCellExperiment (SCE)* class, which is an extension of the *RangedSummarizedExperiment* class. For more details see the package's vignette or refer to the R Documentation.

1.7.1 quality control

The data have been quality controlled using *scPipe* package. It is a Bioconductor package that can handle data generated from all 64 popular 3' end scRNA-seq protocols and their variants (Tian and Su, 2018).

1.7.2 overview

we now stratify the cell and gene data by cell line in each protocol.

```r
## make a summary of QC'ed cell line data processed by each protocol
sce10xqc_smr =  summary(as.factor(sce_sc_10x_qc$cell_line))
sce4qc_smr =    summary(as.factor(sce_sc_CELseq2_qc$cell_line))
scedropqc_smr = summary(as.factor(sce_sc_Dropseq_qc$cell_line))
```

Table 1.1: Summary of cell and gene data per cell line for each protocol

	H1975	H2228	HCC827	Total Cells	Total Genes
10X	313	315	274	902	16468
CEL-seq2	114	81	79	274	28204
Drop-seq	92	65	68	225	15127

```r
## combine the summaries
celline_smr = rbind(sce10xqc_smr,sce4qc_smr,scedropqc_smr)
## produce a 'total' row as well
celline_smr = cbind(celline_smr, apply(celline_smr,1,sum))
## add the genes as well
celline_smr = cbind(celline_smr,
                    c(dim(counts(sce_sc_10x_qc))[1],
                      dim(counts(sce_sc_CELseq2_qc))[1],
                      dim(counts(sce_sc_Dropseq_qc))[1]))
## label the rows
row.names(celline_smr) = c('10X', 'CEL-seq2', 'Drop-seq')
colnames(celline_smr) = c('H1975', 'H2228', 'HCC827',
                          'Total Cells','Total Genes')
```

```r
## tabulate the summaries
kable(celline_smr,
      caption = 'Summary of cell and gene data
      per cell line for each protocol')
```

The assignment to each cell line was performed computationally based on the correlation of the gene expression data with a bulk assay of a mixture of cells by DE analysis using *edgeR*.

The 10X protocol yielded the highest number of cellls (~900). We can visualise the the total amount of genes in each protocol and the amount of overlapping genes among the protocols using a venn diagram:

```r
## create venn diagram of genes in each protocol:
venn.plot = venn.diagram(
  x = list(Chrom.10X = rownames(sce_sc_10x_qc),
           CEL.seq2 = rownames(sce_sc_CELseq2_qc),
           Drop.seq = rownames(sce_sc_Dropseq_qc)),
  filename = NULL, label=T, margin=0.05,
  height = 1400, width = 2200,
  col = 'transparent', fill = c('cornflowerblue','green', 'red'),
  alpha = 0.60, cex = 2, fontfamily = 'serif', fontface = 'bold',
  cat.col = c('darkblue', 'darkgreen', 'black'), cat.cex = 2.2)

## save the plot, change to your own directory
png(filename = 'figures/GeneVenn.png')
grid.draw(venn.plot)
dev.off()
```

A total of 13757 genes are shared across three datasets.

1.7.3 normalisation

Normalisation is typically required in RNA-seq data analysis prior to downstream analysis to reduce heterogeneity among samples due to technical artefact as well as to help to impute the missing values using known

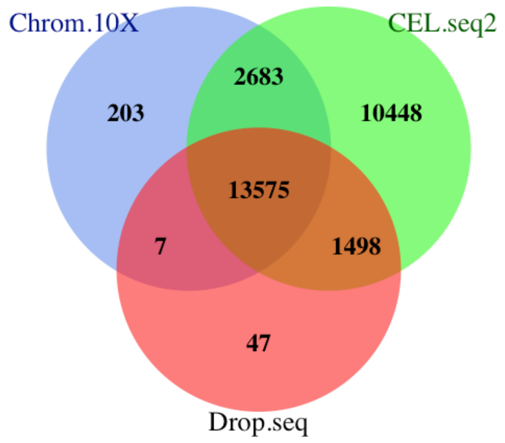


Figure 1.2: The venn diagram of common genes among datasets. MINT uses the common features of all datasets for discriminant analysis.

data.

We use the *scran* package's "Normalisation by deconvolution of size factors from cell pools" method (Lun et al., 2016). It is a two-step process, in which the size factors are adjusted, and then the expression values are normalised:

```
## normalise the QC'ed count matrices
sc10x.norm =  computeSumFactors(sce_sc_10x_qc) ## deconvolute using size factors
sc10x.norm =  normalize(sc10x.norm) ## normalise expression values
## DROP-seq
scdrop.norm = computeSumFactors(sce_sc_Dropseq_qc)
scdrop.norm = normalize(scdrop.norm)
## CEL-seq2
sccel.norm =  computeSumFactors(sce_sc_CELseq2_qc)
sccel.norm =  normalize(sccel.norm)
```

Depending on the data structure and/or the question in hand the following analyses can have one of the following two forms:

- **Unsupervised Analysis:** Where we want to explore the patterns among the data and find the main directions that drive the variations in the dataset. **It is often beneficial to perform unsupervised analysis prior to supervised analysis to assess the presence of outliers and/or batch effects**.
- **Supervised Analysis:** In which each sample consists of a pair of feature measurements (e.g. gene counts) and class membership (e.g. cell type). The aim is to build a model that maps the measurements to their classes using a training set which can adequately predict classes in a test set as well.

We will start with Unsupervised Analysis and then proceed to Supervised Analysis.

1.8 PCA: unsupervised analysis

We start by Principal Component Analysis (PCA) (Jolliffe, 1986). It is a dimension reduction method which seeks for components that maximise the variance within the datasets. PCA is primarily used to explore one single type of 'omics data (e.g. transcriptomics, proteomics, metabolomics data, etc). It is an unsupervised learning method and thus there is no assumption of data corresponding to any class (batch, study etc).

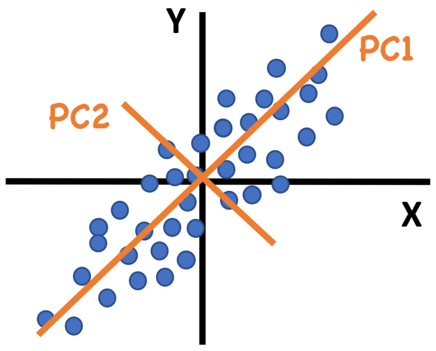


Figure 1.3: An example of principal components for data with only 2 dimensions. The first PC is along the direction of maximum variance.

The first Principal Component (PC) explains as much variance in the data as possible, and each following PC explains as much of the remaining variance as possible, in a direction orthogonal to all previous PCs. Thus, the first few components often capture the main variance in a dataset.

We will perform PCA on the datasets from each protocol separately and then conduct a PCA on the concatenated dataset.

1.8.1 PCA on each protocol individually

We will use *mixOmics*'s *pca* function in this section. The method is implemented numerically and it is capable of dealing with datasets with or without missing values (see documentation for details). The *pca* function takes the data as a matrix of count data. Contrary to conventional biological data formation, *pca* takes each row to be the gene expression profile of a molecule/cell. Therefore, we will have to transpose the normalised matrix (using the *t()* function) to perform PCA. Additionally, we use the log transformed counts (*logcounts()*) to confine the counts' span for numerical reasons.

By default, *mixOmics* centres the normalised matrix to have zero mean. The *scale* argument can also be set to 'TRUE' if there are diverse units in raw data.

The *ncomp* argument denotes the number of desired PCs to find[1]. We will retrieve 10 PCs for each protocol at this stage:

```r
## pca on the normalised count matrices and find 10 PCs
pca.res.10x =     pca(t(logcounts(sc10x.norm)),  ncomp = 10,
                      center=T, scale=F)
pca.res.celseq =  pca(t(logcounts(sccel.norm)),  ncomp = 10,
                      center=T, scale=F)
pca.res.dropseq = pca(t(logcounts(scdrop.norm)), ncomp = 10,
                      center=T, scale=F)
```

Each output is a *pca* object that includes the centred count matrix for that protocol, the mean normalised counts for each gene, the PCA loadings, and score values.

It is possible to use the *plot* function on *pca* object to visualise the proportion of the total variance of the data explained by each of the 10 PCs for each protocol using a barplot:

```r
## arrange the plots in 1 row and 3 columns
par(mfrow=c(1,3))
## find the maximum explained variance in all PCs:
ymax = max (pca.res.10x$explained_variance[1],
```

[1]More arguments: *max.iter, tol, logratio, multilevel.*

```
             pca.res.celseq$explained_variance[1],
             pca.res.dropseq$explained_variance[1])

## plot the pca objects and limit the Y axes to ymax for all
plot(pca.res.10x, main= '(A) 10X', ylim=c(0,ymax))
plot(pca.res.celseq,  main= '(B) CEL-seq2', ylim=c(0,ymax))
plot(pca.res.dropseq,  main= '(C) Drop-seq', ylim=c(0,ymax))
```

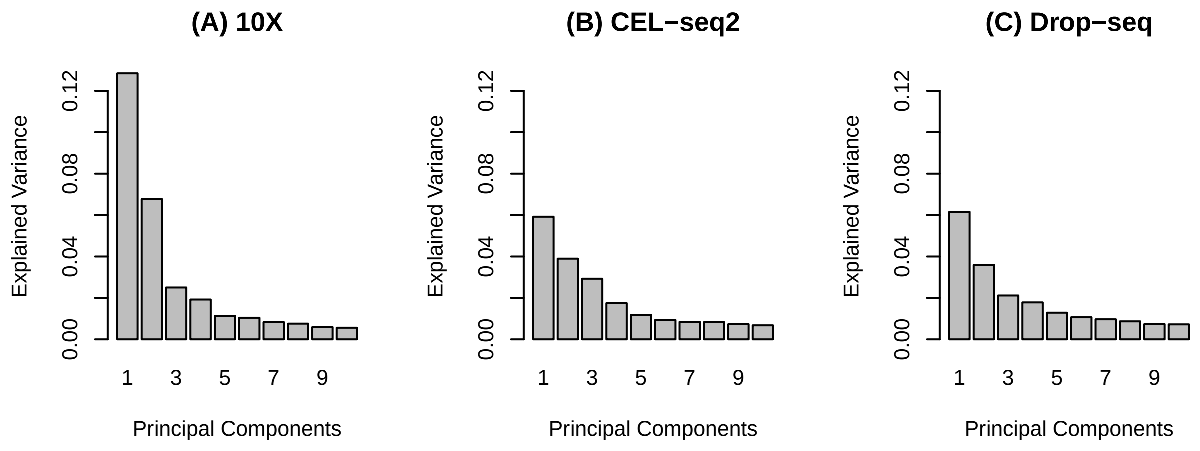


Figure 1.4: The pca barplot for each protocol. (A) The first 2 PCs explain 20% of total variability of the data and there is a drop (elbow) in the explained variability afterwards. (B) The first 2 PCs explain 10% of variability and an elbow is not apparent. (C) similar cumulative explained variance to CEL-seq2 for first 2 PCs.

It will usually be desirable if the first few PCs capture sufficient variation in the data, as this will help with visualisation.

In order to have a simple 2D scatter plot, the first 2 PCs are usually the ones of interest in PCA plots[2]. Such a plot can be created using the *plotIndiv* function. For a complete list of the argument options refer to the documentation.

We will colour the data points of each cell line using *group* argument to see whether there is a differentiation between different cell types:

```
## define custom colours/shapes
## colour by cell line
col.cell = c('H1975'='#0000ff', 'HCC827'='grey30', 'H2228' ='#ff8000')
## shape by batch
shape.batch = c('10X' = 1, 'CEL-seq2'=2, 'Drop-seq'=3 )
```

```
## pca plots for protocols
## 10x
plotIndiv(pca.res.10x, legend = T, title = 'PCA 10X', pch = shape.batch['10X'], col = col.cell,
          group = sce_sc_10x_qc$cell_line, legend.title = 'Cell Line')
```

PCA plot for 10X data shows that the H2228 cells are most differentiated from others along the first PC, while the other two are located similarly along this PC. The 3 clusters are separated along the second PC. The Drop-seq data have been quality controlled twice and still exhibit two clusters. One possible explanation

[2]Using the “style=‘3d’ ” argument, one can also plot the 3d PCA plot with 3 components

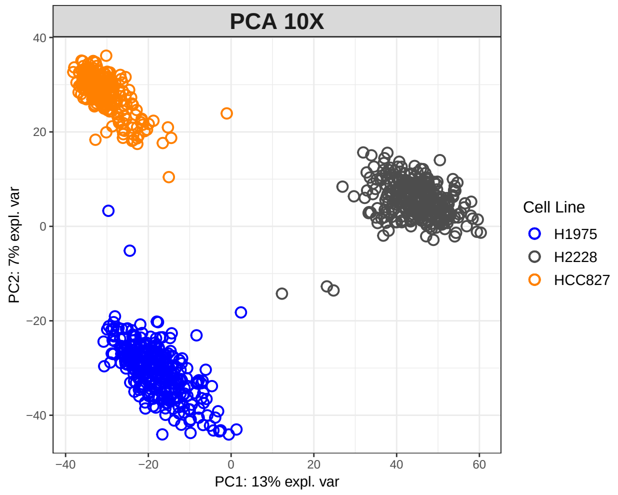


Figure 1.5: PCA plot for the 10X dataset. The data tend to group together by cell lines.

is the existence of doublets (droplets with 2 cells). This highlights the importance of carefully tuning the experiment parameters.

```r
## CEL-seq2
plotIndiv(pca.res.celseq, legend = T, title = 'PCA CEL-seq2', pch = shape.batch['CEL-seq2'],
          col = col.cell, group = sce_sc_CELseq2_qc$cell_line, legend.title = 'Cell Line')
```

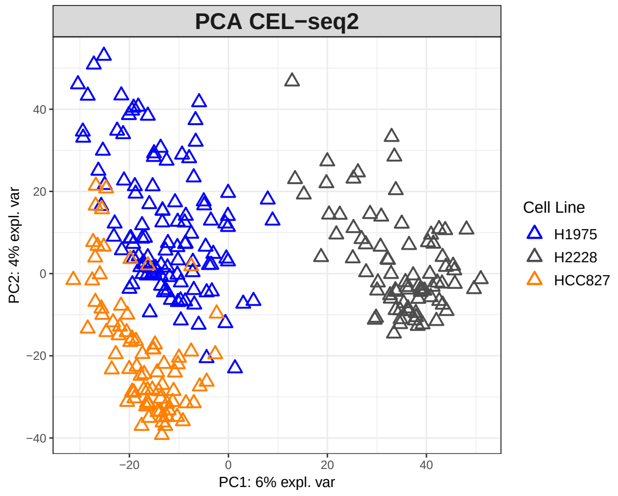


Figure 1.6: The PCA plots for the CEL-seq2 data. The data are widely scattered in the 2D plane while H2228 cells are relatively distant from the other two.

```r
## Drop-seq
plotIndiv(pca.res.dropseq, legend = T, title = 'PCA Drop-seq', pch = shape.batch['Drop-seq'],
          col = col.cell, group = sce_sc_Dropseq_qc$cell_line, legend.title = 'Cell Line')
```

1.9 PCA on the combined dataset

We now pool/concatenate the datasets for unsupervised analysis. First we should find out the common genes across the three datasets, as a requirement for **P-Integration**:

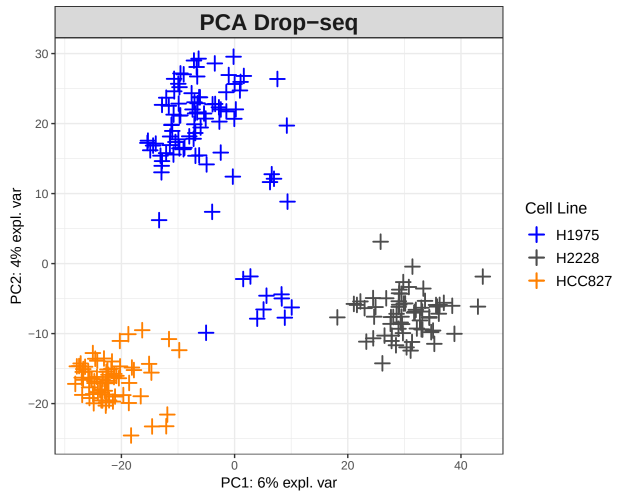


Figure 1.7: The PCA plots for the Drop-seq data. While H2228 and HCC827 cell data tend to cluster by cell line, the H1975 data exhibit two clusters with negative correlation (on opposite sides of origin). The within-data variation is not consistent between datasets. For instance, the 10X data show a clear grouping structure by cell lines, while this observation is not as strongly supported in other datasets.

```
## find the intersect of the genes for integration
list.intersect = Reduce(intersect, list(
## the rownames of the original (un-transposed) count matrix will -
## output the genes
  rownames(logcounts(sc10x.norm)),
  rownames(logcounts(sccel.norm)),
  rownames(logcounts(scdrop.norm))
))
```

Now we can merge the 3 datasets:

```
## combine the data at their intersection
data.combined = t( ## transpose of all 3 datasets combined
  data.frame(
    ## the genes from each protocol that match list.intersect
    logcounts(sc10x.norm)[list.intersect,],
    logcounts(sccel.norm)[list.intersect,],
    logcounts(scdrop.norm)[list.intersect,] ))
## the number of cells and genes in the intersect dataset
dim(data.combined)
```

```
## [1]  1401 13575
```

This matrix includes the count data for the combined dataset. We will also create 2 vectors of the cell lines and batches for visualisation of the PCA plots, and also later for the PLS-DA analysis:

```
## create a factor variable of cell lines
## must be in the same order as the data combination
cell.line = as.factor(c(sce_sc_10x_qc$cell_line,
                         sce_sc_CELseq2_qc$cell_line,
                         sce_sc_Dropseq_qc$cell_line))
## name the factor variable with the cell ID
names(cell.line) = rownames(data.combined)
```

```r
## produce a character vector of batch names
## must be in the same order as the data combination
batch = as.factor(
  c(rep('10X',      ncol(logcounts(sc10x.norm))),
    rep('CEL-seq2',  ncol(logcounts(sccel.norm))),
    rep('Drop-seq', ncol(logcounts(scdrop.norm))) ))
## name it with corresponding cell IDs
names(batch) = rownames(data.combined)
```

We can now perform PCA and visualise the results:

```r
## perform PCA on concatenated data and retrieve 2 PCs
pca.combined = pca(data.combined, ncomp = 2)
```

```r
## plot the combined pca coloured by batches
plotIndiv(pca.combined, title = 'PCA Combined',
          pch = batch, ## shape by cell line
          group = cell.line, ## colour by batch
          legend = T, legend.title = 'Cell Line',
          legend.title.pch = 'Study')
```

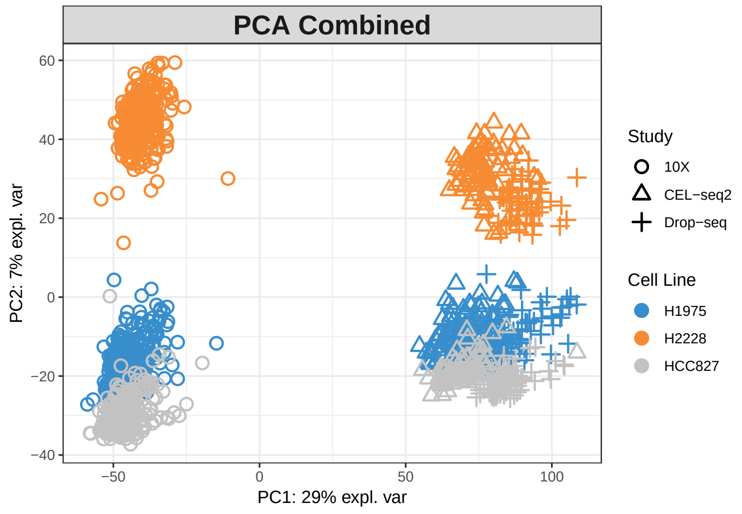


Figure 1.8: The PCA plot for the combined data, coloured by cell lines.

```r
## plot the combined pca coloured by protocols
plotIndiv(pca.combined, title = 'PCA Combined',
          pch = cell.line, ## shape by cell line
          group = batch, ## colour by protocol
          col.per.group = c('red', 'purple', 'green'),
          legend = T, legend.title = 'Study',
          legend.title.pch = 'Cell Line')
```

As shown in combined PCA plots above, the protocols are driving the variation along PC1 (batch effects), while wanted biological variation is separating the data along PC2. We will next implement the MINT PLS-DA method on the combined dataset with an aim to account for the batch effects in discriminating the different cell types.

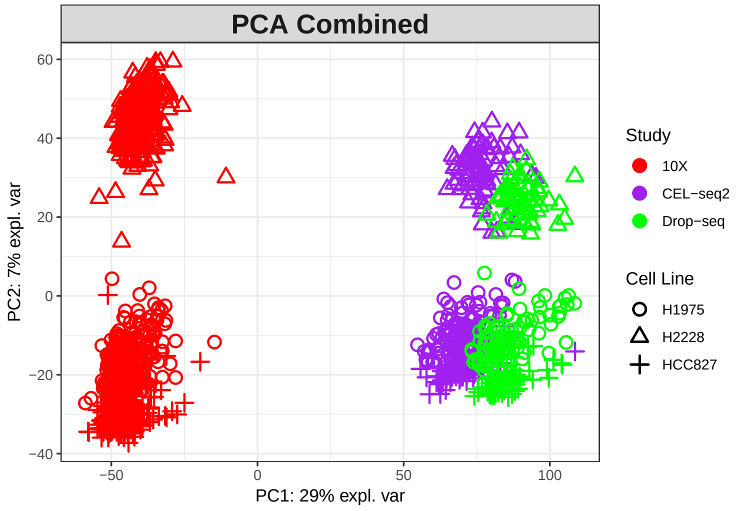


Figure 1.9: The PCA plot for the combined data, coloured by protocols.

1.10 MINT to combine the datasets

MINT uses **Projection to Latent Structures - Discriminant Analysis** to build a learning model on the basis of a training dataset. Such a predictive model is expected to accurately classify the samples with unknown cell types in an external learning dataset.

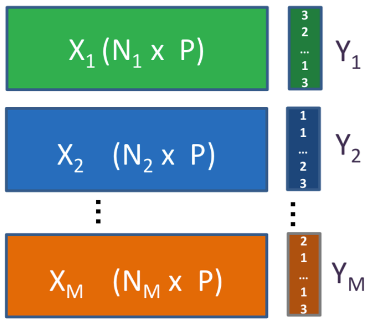


Figure 1.10: P-integration framework using MINT for M independent studies (X) on the same P features. In this setting Y is a vector indicating the class of each cell.

1.10.1 method

MINT finds a set of discriminative latent variables (in this context, **linear combinations of gene expression values**) simultaneously in all the datasets, thus leading to PLS-DA Components (as opposed to PCs in PCA) most influenced by consistent biological heterogeneity across the studies. Therefore, these components do not necessarily maximise the variance among the pooled data like what PCs do, but maximise covariance between the combined data and their classes (cell lines).

The *mint.plsda* function in *mixOmics* has a set of inputs to perform the analysis, including:

- **X**: Which is the original predictor matrix.
- **Y**: Factor of classes (here, cell lines).
- **study**: Factor indicating the membership of each sample to each of the studies/batches being combined. For a detailed list of functions available with *MINT* refer to the documentation.

Since PLS-DA is a supervised method, we initially create a vector to assign each sample to its class/cell.line (Y) and then perform *MINT*, keeping 5 PLS-DA components:

```r
## create variables needed for MINT
## factor variable of cell lines
Y = as.factor(cell.line[rownames(data.combined)])
## factor variable of studies
study = batch ## defined in the combined PCA section
## MINT on the combined dataset with 5 components
mint.plsda.res = mint.plsda(X = data.combined, Y = Y,
                            study = study, ncomp = 5)
```

The outcome is a *mint.plsda* object which can be plotted using *plotIndiv* function:

```r
## plot the mint.plsda plots for the combined dataset
plotIndiv(mint.plsda.res, group = cell.line,
          legend  = T, subtitle     = 'MINT - Coloured by Cell Line',
          ellipse = F, legend.title = 'Cell Line',
          legend.title.pch = 'protocol',
          X.label = 'PLS-DA component 1',
          Y.label = 'PLS-DA component 2')
```

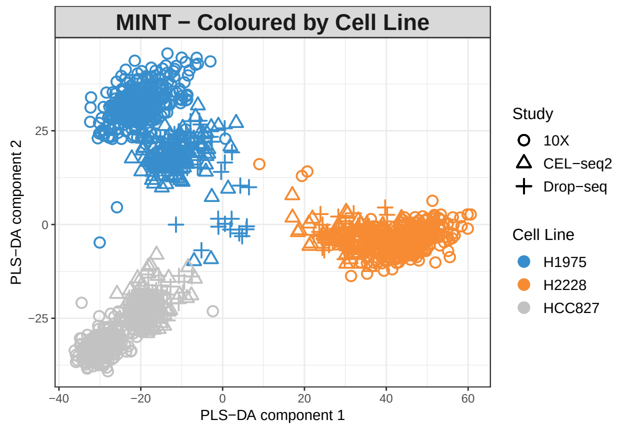


Figure 1.11: The MINT PLS-DA plot for the combined dataset. Data points are coloured by cell lines.

As seen in the PLS-DA plots, the data are differentiated mainly by their cell lines. The effects of batches are not fully eliminated yet.

1.10.2 optimum number of components

We can look at the calassification error rates over the chosen number of components (5) using *perf* function in *mixOmics*, which evaluates the performance of the fitted PLS models internally using Leave-One-Group-Out Cross Validation (LOGOCV) in *MINT* (M-Fold Cross-Validation is also available). Refer to the documentation for more details about *perf* arguments.

```r
## perform cross validation and calculate classification error rates
set.seed(12321)  # for reproducibility of the results
perf.mint.plsda.res = perf(mint.plsda.res,
          progressBar = F)
```

We now plot the output:

```r
## plot the classification error rate vs number of components
plot(perf.mint.plsda.res, col = color.mixo(5:7))
```

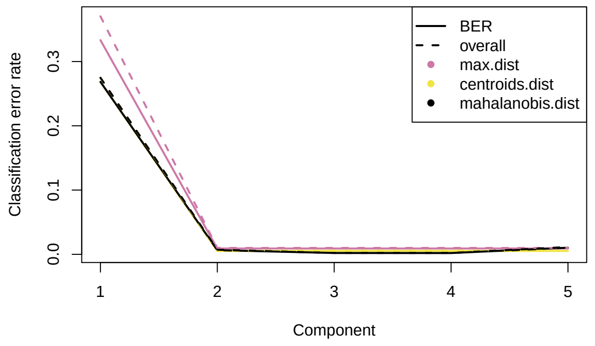


Figure 1.12: The classification error rate for different number of PLS-DA components showing Balanced and Overall Error Rates, each comprising three distance measures. The distances measure how far any given data point is from the mean of its class.

As seen in the plot above, at ncomp = 2 the model has the optimum performance for all distances in terms of Balanced and Overall Error Rate, as there is not a considerable drop in error rates for further components (see supplemental information from (Le Cao et al., 2017) for more details about the prediction distances). Additional numerical outputs are available to stratify the error rates per cell line/protocol/distance measure:

```r
perf.mint.plsda.res$global.error$BER ## further error diagnostics
```

```
##           max.dist centroids.dist mahalanobis.dist
## comp 1 0.333333333    0.268431385      0.268431385
## comp 2 0.009072456    0.005218891      0.006503413
## comp 3 0.009072456    0.005218891      0.002088392
## comp 4 0.009072456    0.005218891      0.002007587
## comp 5 0.009072456    0.005218891      0.010356977
```

Also, the function outputs the optimal number of components via the *choice.ncomp* attribute.

```r
## number of variables to select in each component
perf.mint.plsda.res$choice.ncomp
```

```
##         max.dist centroids.dist mahalanobis.dist
## overall        2              2                2
## BER            2              2                2
```

1.10.3 sparse MINT PLS-DA

At this section, we investigate the performance of *MINT* with variable selection (we call it sparse MINT PLSDA or MINT sPLS-DA). This helps by eliminating the noisy variables (here genes) from the model to better interpret the signal coming from the primary variables. At this stage, we will ask the model to keep 50 variables with highest loadings on each component. The number of variables to keep should be set as a vector into *keepX* argument in *mint.splsda* (refer to the documentation for more details):

```r
## number of variables to keep in each component
list.keepX = c(50,50)
## perform sparse PLS-DA using MINT with 2 components
```

```
mint.splsda.res = mint.splsda(X = data.combined, Y = Y,
                              study = study, ncomp = 2, keepX = list.keepX)
```

Plot the sparse *MINT* object:

```
plotIndiv(mint.splsda.res,
          legend  = T, subtitle = 'Sparse MINT', ellipse = T,
          X.label = 'sPLS-DA component 1',
          Y.label = 'sPLS-DA component 2',
          group = Y, ## colour by cell line
          legend.title = 'Cell Line',
          legend.title.pch = 'Protocol')
```

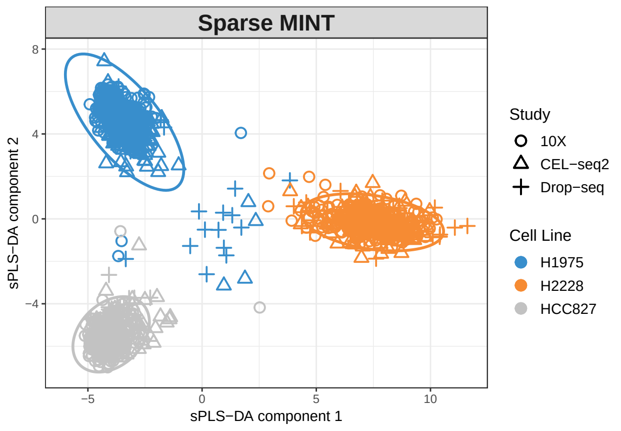


Figure 1.13: The Sparse MINT PLS-DA plot for the combined dataset. Data points are coloured by cell lines.

The clusters are now more refined compared to the non-sparse method. The effects of batches are now almost removed, and the samples from the 10X dataset seem equally differentiated as the others.

1.10.4 choice of parameters

Previously, we chose 50 variables for a sparse analysis. The optimal number of the variables can be determined for each component using the *tune* function. The *test.keepX* argument specifies a vector of the candidate numbers of variables in each PLS-DA component to be evaluated. Here, we will only try 5,10,...,35 variables to save on time, and also record the runtime. It is recommended to use a coarse and wide grid first (e.g. seq(10,300,10)) and refine it once the right neighbourhood is known. We will separately tune each component for better visualisation but tuning can be done in one step for all components.

```
## tune MINT for component 1 and then 2 and record the run time
## we tune individual components for visualisation purposes
## one can run only the tune.mint.c2 without already.tested.X
start.time = Sys.time()
## tune using a test set of variable numbers
## component 1
tune.mint.c1 = tune(
  X = data.combined, Y = Y, study = study, ncomp = 1,
  ## assess numbers 5,10,15...35:
  test.keepX = seq(5,35,5), method = 'mint.splsda',
  ## use all distances to estimate the classification error rate
  dist = c('max.dist',  'centroids.dist', 'mahalanobis.dist'),
```

```r
  progressBar = F
)
## component 1 to 2
tune.mint.c2 = tune(
  X = data.combined, Y = Y, study = study, ncomp = 2,
  ## already tuned component 1
  already.tested.X = tune.mint.c1$choice.keepX,
  test.keepX = seq(5,35,5), method = 'mint.splsda',
  dist = c('max.dist',  'centroids.dist', 'mahalanobis.dist'),
  progressBar = F
)
end.time = Sys.time()
## see how long it takes to find the optimum number of variables:
run.time =  end.time - start.time
```

It took less than 3 minutes to evaluate the chosen test set of variables using a 2.6 GHz Dual-core Intel Core i5 processor (8GB RAM). It is always more practical to look into a coarse grid at first and then refine it when the right neighbourhood is found. We now look at the optimum number of components chosen and their corresponding Balanced Error Rate:

```r
## look at the optimal selected variables for each PC
tune.mint.c2$choice.keepX
```

```
## comp 1 comp 2
##     35     10
```

```r
## plot the error rates for all test variable numbers
par(mfrow=c(1,3))
plot(tune.mint.c1, col = 'darkred')
plot(tune.mint.c2, col = 'darkblue')
```

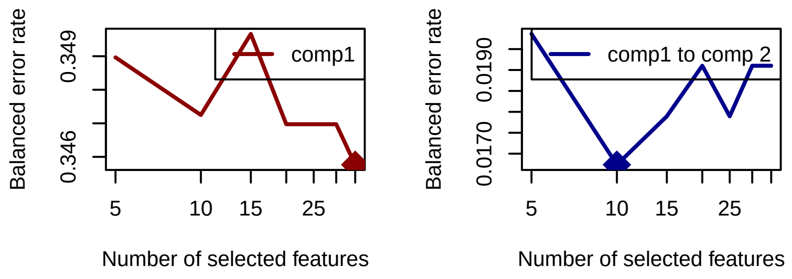


Figure 1.14: The Balanced Error Rate as a function of number of variables in PLS-DA components 1 (left) and 2 (right).

The optimum numbers of variables for each component are shown using a diamond mark in plots above, which are 35 for the first and 10 for the second one.

We now re-run the sparse *MINT* using optimum parameters:

```r
## run sparse mint using optimum parameters:
mint.splsda.tuned.res = mint.splsda( X =data.combined, Y = Y,
                              study = study, ncomp = 2,
                        keepX = tune.mint.c2$choice.keepX)
```

Next, we plot the *mint.splsda* object with global variables (for the combined dataset):

```
## plot the tuned mint.splsda plot for the combined dataset
plotIndiv(mint.splsda.tuned.res, study = 'global', legend = T,
          title = 'MINT sPLS-DA',  subtitle = 'Global', ellipse=T, legend.title = 'Cell Line')
```

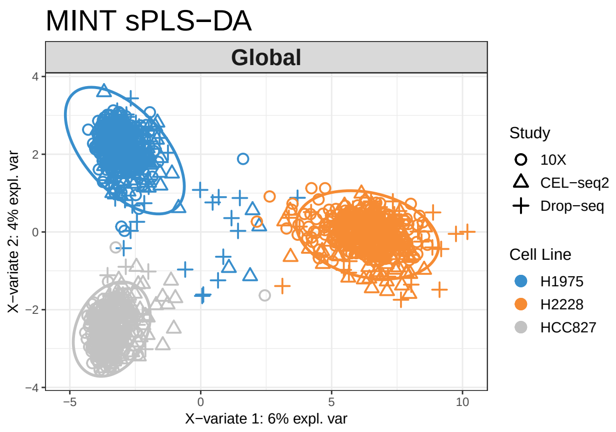


Figure 1.15: The tuned MINT sPLS-DA plot for the combined data. While there are still a number of samples that are not well classified, the clusters are more refined when we perform variable selection.

We can also look at the dataset per protocol using *all.partial* as the input for the *study* argument:

```
## tuned mint.splsda plot for each protocol
plotIndiv(mint.splsda.tuned.res, study = 'all.partial',  title = 'MINT sPLS-DA',
          subtitle = c('10X', 'CEL-seq2', 'Drop-seq'))
```

The majority of samples from the 10X data are well classified, while the method struggles to classify some samples from H1975 samples in the Drop-seq and CEL-seq2 datasets.

1.10.5 performance assessment

We now can evaluate the classification performance of the final model using *perf* function. We will use the maximum distance measure.

```
set.seed(12321)  # for reproducibility of the results
## perform classification with leave-one-group-out cross validation
perf.mint.final = perf(mint.splsda.res, progressBar = F, dist = 'max.dist')
## classification error rate
perf.mint.final$global.error
```

```
## $BER
##          max.dist
## comp 1 0.35106644
## comp 2 0.01156069
##
## $overall
##          max.dist
## comp 1 0.33333333
## comp 2 0.01284797
##
```

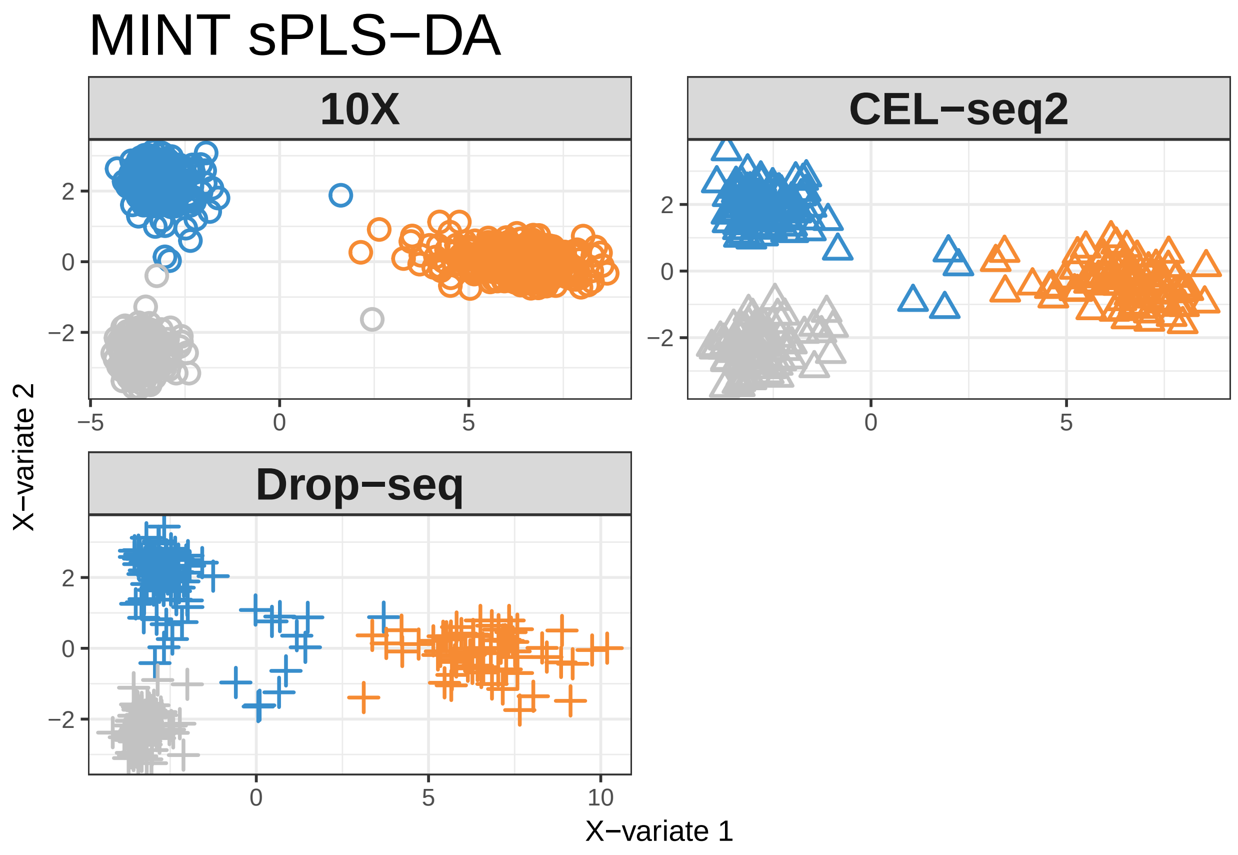


Figure 1.16: MINT sPLS-DA components for each individual protocol coloured by cell line.

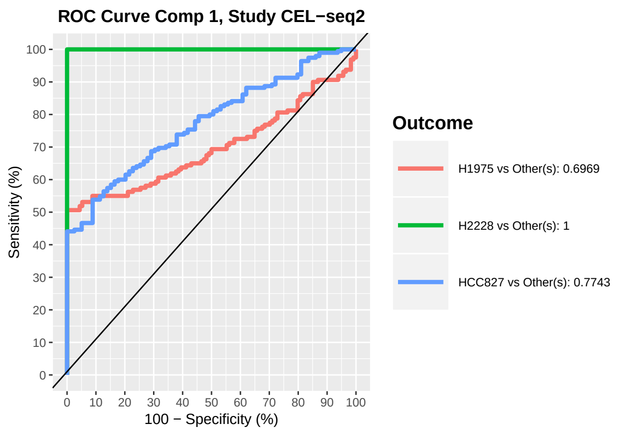
```
## $error.rate.class
## $error.rate.class$max.dist
##            comp 1     comp 2
## H1975  0.2408478 0.03468208
## H2228  0.0000000 0.00000000
## HCC827 0.8123515 0.00000000
```

The balanced and overall error rates are 1.2 and 1.3 percent respectively, while the model classifies the H2228 cell types with zero error rate.

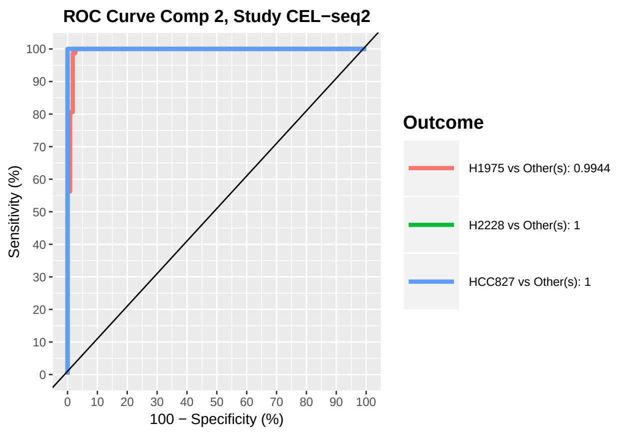
1.10.6 ROC curves

Another available visualisation tool is the ROC (Receiver Operating Characteristic) curves for each study (or all) to evaluate the prediction accuracy of a classification mode. In a ROC curve the true positive rate (Sensitivity) is plotted against the false positive rate (100-Specificity) for different classification thresholds. This method should be interpreted carefully in such multivariate analysis where no clear cut-off threshold can be considered.

```
## ROC curves for both components
auc.mint.splsda1 = auroc(mint.splsda.tuned.res, roc.comp = 1, roc.study='CEL-seq2')
```



```
auc.mint.splsda2 = auroc(mint.splsda.tuned.res, roc.comp = 2, roc.study='CEL-seq2')
```



For a perfect model the curve would go through the upper left corner, while the black line represents a perfectly random classification model.

ROC plots show that for CEL-seq data, the model has relatively low prediction accuracy for HCC827 and H1975 samples in the first component, while it has high prediction accuracy for all samples in the second component. This was also apparent in sPLS-DA plot, as HCC827 and H1975 samples were not separated along the first PLSDA component for CEL-seq data, while most of samples were differentiated along the second component by their cell lines.

1.11 session information

```
writeLines(capture.output(sessionInfo()), "sessionInfo.md")
## session information to build this vignette
sessionInfo()
```

```
## R version 3.5.0 (2018-04-23)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS  10.14.1
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
```

```
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
##  [1] grid      parallel  stats4    stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] tibble_1.4.2                VennDiagram_1.6.20
##  [3] futile.logger_1.4.3         scran_1.9.39
##  [5] mixOmics_6.7.0              ggplot2_3.1.0
##  [7] lattice_0.20-38             MASS_7.3-51.1
##  [9] SingleCellExperiment_1.3.12 SummarizedExperiment_1.11.6
## [11] DelayedArray_0.7.49         BiocParallel_1.15.15
## [13] matrixStats_0.54.0          Biobase_2.41.2
## [15] GenomicRanges_1.33.14       GenomeInfoDb_1.17.4
## [17] IRanges_2.15.19             S4Vectors_0.19.24
## [19] BiocGenerics_0.27.1         knitr_1.20
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-6              RColorBrewer_1.1-2
##  [3] rprojroot_1.3-2           dynamicTreeCut_1.63-1
##  [5] tools_3.5.0               backports_1.1.2
##  [7] R6_2.3.0                  HDF5Array_1.9.19
##  [9] vipor_0.4.5               lazyeval_0.2.1
## [11] colorspace_1.3-2          withr_2.1.2
## [13] tidyselect_0.2.5          gridExtra_2.3
## [15] compiler_3.5.0            BiocNeighbors_0.99.22
## [17] formatR_1.5               labeling_0.3
## [19] bookdown_0.7              scales_1.0.0
## [21] stringr_1.3.1             digest_0.6.18
## [23] rmarkdown_1.10            XVector_0.21.4
## [25] scater_1.9.24             pkgconfig_2.0.2
## [27] htmltools_0.3.6           limma_3.37.10
## [29] rlang_0.3.0.1             DelayedMatrixStats_1.3.11
## [31] bindr_0.1.1               dplyr_0.7.8
## [33] RCurl_1.95-4.11           magrittr_1.5
## [35] GenomeInfoDbData_1.2.0    Matrix_1.2-15
## [37] Rcpp_1.0.0                ggbeeswarm_0.6.0
## [39] munsell_0.5.0             Rhdf5lib_1.3.3
## [41] viridis_0.5.1             stringi_1.2.4
## [43] yaml_2.2.0                edgeR_3.23.7
## [45] zlibbioc_1.27.0           rhdf5_2.25.11
## [47] plyr_1.8.4                crayon_1.3.4
## [49] locfit_1.5-9.1            pillar_1.3.0
## [51] igraph_1.2.2              corpcor_1.6.9
## [53] reshape2_1.4.3            codetools_0.2-15
## [55] futile.options_1.0.1      glue_1.3.0
## [57] evaluate_0.12             lambda.r_1.2.3
## [59] BiocManager_1.30.4        gtable_0.2.0
## [61] purrr_0.2.5               tidyr_0.8.2
## [63] assertthat_0.2.0          xfun_0.4
## [65] RSpectra_0.13-1           viridisLite_0.3.0
## [67] rARPACK_0.11-0            beeswarm_0.2.3
## [69] ellipse_0.4.1             bindrcpp_0.2.2
## [71] statmod_1.4.30
```

1.12 troubleshoots

1.12.1 Bioconductor

You can check out Bioconductor's troubleshoot page

1.12.2 mixOmics

In case you are having issues with *mixOmics*, please look it up at the mixOmics issues page and if you did not find your solution, create a new issue. You can also check out or submit your questions on stackoverflow forums.

1.12.3 mac OS

- **Compilation error while installing libraries from binconductor** This could be due to the fortran compiler not being updated for newer R versions. You can go to this website and get the instructions on how to install the latest gfortran for your R and mac OS version.
- **Unable to import *rgl* library** Ensure you have the latest version of XQuartz installed.

Chapter 2

Identifying A Gene Signature Using MINT

2.1 prerequisites

This section follows up on the Data Integration vignette. The aim here is to find the gene signature characterising the cell lines using the MINT sPLS-DA model and also to make relevant comparisons.

The following topics are covered in details in the previous vignette and thus will not be expanded:

- The instructions on installation of the required packages
- Data normalisation method using *scran* package

2.2 R scripts

The R scripts are available at this link.

2.3 libraries

```r
## load the required libraries
library(SingleCellExperiment)
library(mixOmics)
library(scran)
library(knitr)
library(VennDiagram)
library(tibble)
```

2.4 data

Here we set up the input and output specified in *params* section of *yaml*. By default, all required data are loaded from GitHub and run data are not saved.

```r
## check for valid input/output setup
check.exists <- function(object) ## function to assess existence of objects
{
  exists(as.character(substitute(object)))
}
```

```r
## input/output from parameters
io = list()

## whether or where from to locally load data - FALSE: GitHub load; or a directory
io$local.input = ifelse(check.exists(params$local$input), params$local.input, F)

## whether or where to save run data - FALSE: do not save; or a directory
io$output.data = ifelse(check.exists(params$output.data), params$output.data, F)

## whether or where to save R scripts - FALSE: do not save; or a directory
io$Rscripts=ifelse(check.exists(params$Rscripts), params$Rscripts, F)

## whether the normalised sce and mint objects from output.data should be re-calculated (TRUE) or direc
io$recalc = ifelse(check.exists(params$recalc), params$recalc, F)
```

The *sce* objects can be re-load and normalised for reproducibility of the vignette. You may also load the normalised data and skip these steps (refer to *input/output* section):

```r
if (isFALSE(io$local.input)&&io$recalc){
  ## load from GitHub
  DataURL='https://tinyurl.com/sincell-with-class-RData-LuyiT'
  load(url(DataURL))
} else if (!isFALSE(io$local.input)&&io$recalc){
  load(file.path(io$local.input, 'sincell_with_class.RData'))
  }
```

```r
if (io$recalc){
  ## normalise the QC'ed count matrices
  sc10x.norm =  computeSumFactors(sce_sc_10x_qc) ## deconvolute using size factors
  sc10x.norm =  normalize(sc10x.norm) ## normalise expression values
  ## DROP-seq
  scdrop.norm = computeSumFactors(sce_sc_Dropseq_qc)
  scdrop.norm = normalize(scdrop.norm)
  ## CEL-seq2
  sccel.norm =  computeSumFactors(sce_sc_CELseq2_qc)
  sccel.norm =  normalize(sccel.norm)
} else {
  ## load locally - change to your own
  load('../output/sce.norm.RData')
}
```

2.5 PLS-DA on each protocol individually

Previously, we combined the datasets using MINT sPLS-DA. We then performed variable selection using optimum number of variables for each MINT PLS-DA component to tune the model parameters using key predictors. In this vignette, we will also carry out (s)PLS-DA on each individual dataset (from every protocol) to compare the signatures from individual and combined studies. Next, we will compare the signatures against differentially expressed genes from a univariate analysis from CellBench study.

2.5.1 method

As mentioned in previous vignette, PLS-DA finds the molecular signature that drives the association of cells to their cell lines. Here we use mixOmics' **plsda** function to find the PLSDA components in each dataset, refer to the documentation for details.

Since our aim is to find the signature from each dataset and compare, we need to find the data corresponding to common genes across all datasets:

```r
## find the intersect of the genes
list.intersect = Reduce(intersect, list(
## the rownames of the original (un-transposed) count matrix will output the genes
  rownames(logcounts(sc10x.norm)),
  rownames(logcounts(sccel.norm)),
  rownames(logcounts(scdrop.norm))
))
```

Initially, we apply the function to the 10X data and perform cross validation:

```r
## PLSDA - 10x
## extract the normalised count matrix from the SCE object (transposed)
normalised.10x = t(logcounts(sc10x.norm))
## keep the common genes only for comparability
normalised.10x = normalised.10x[,list.intersect]
## form a factor variable of cell lines
Y.10x = as.factor(sc10x.norm[list.intersect,]$cell_line)
## PLS-DA on the dataset with 5 components
plsda.10x.res = plsda(X = normalised.10x, Y = Y.10x, ncomp = 5)
## perform cross validation and find the classification error rates
start = Sys.time()
perf.plsda.10x = perf(plsda.10x.res, progressBar=F )
run.time = Sys.time()-start
```

The run took less than 2 mins. We can now plot the error rate profile:

```r
## optimal number of components
plot(perf.plsda.10x, col = color.mixo(5:7))
```

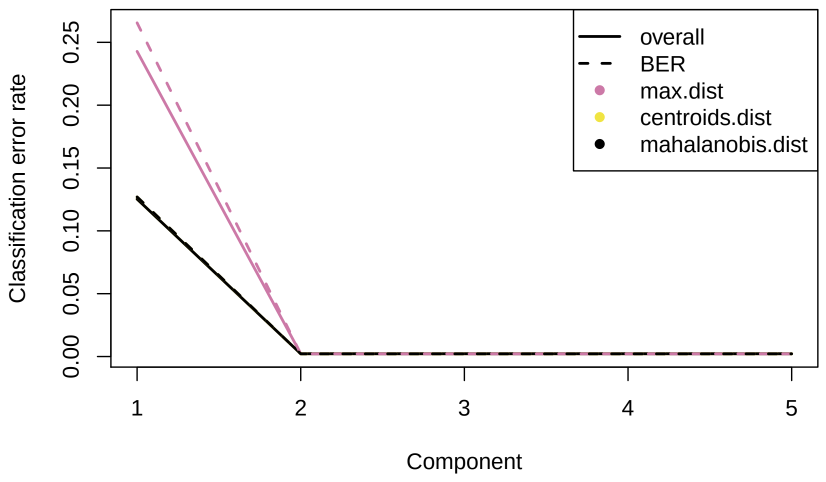


Figure 2.1: The classification error rate for different number of PLSDA components of 10X dataset.

Similar to the combined dataset, *ncomp*=2 leads to optimum error rate. We now apply *plsda* to the other 2 datasets:

```r
## PLSDA - CEL-seq2 and Drop-seq
## extract the normalised count matrix from the SCE object (transposed)
## CEL-seq2
normalised.cel = t(logcounts(sccel.norm))
normalised.cel = normalised.cel[,list.intersect]
## Drop-seq
normalised.drop= t(logcounts(scdrop.norm))
normalised.drop = normalised.drop[,list.intersect]
## factor variable of cell lines
Y.cel = as.factor(sccel.norm[list.intersect,]$cell_line)
Y.drop = as.factor(scdrop.norm[list.intersect,]$cell_line)
## PLS-DA on each dataset with 2 components
plsda.10x.res = plsda(X = normalised.10x, Y = Y.10x, ncomp = 2)
plsda.cel.res = plsda(X = normalised.cel, Y = Y.cel, ncomp = 2)
plsda.drop.res = plsda(X = normalised.drop, Y = Y.drop, ncomp = 2)
```

We plot the plsda plots for the 10X and CEL-seq2 data:

```r
## mint.plsda plot for 10X
plotIndiv(plsda.10x.res,
          legend  = T, title      = 'PLSDA 10X',
          ellipse = T, legend.title = 'Cell Line',
          X.label = 'PLSDA component 1',
          Y.label = 'PLSDA component 2', pch=1)
```

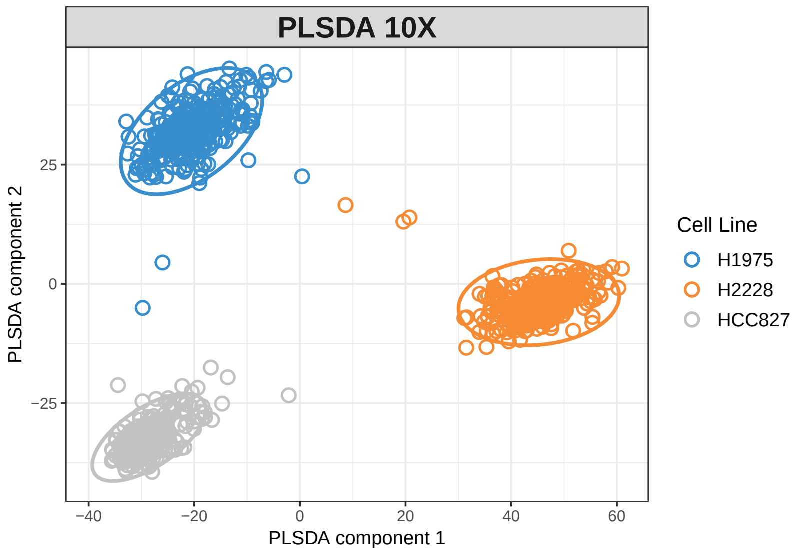


Figure 2.2: PLSDA plot for the 10x dataset.

```r
## mint.plsda plot for CEL-seq2
plotIndiv(plsda.cel.res,
          legend  = T, title      = 'PLSDA CEL-seq2',
          ellipse = T, legend.title = 'Cell Line',
          X.label = 'PLSDA component 1',
          Y.label = 'PLSDA component 2', pch=2)
```

The PLSDA figures show clustering by cell line, while the CEL-seq2 data are more scattered compared to 10X. In interpreting the relations, it is important to note that the apparent change of clusters is simply the

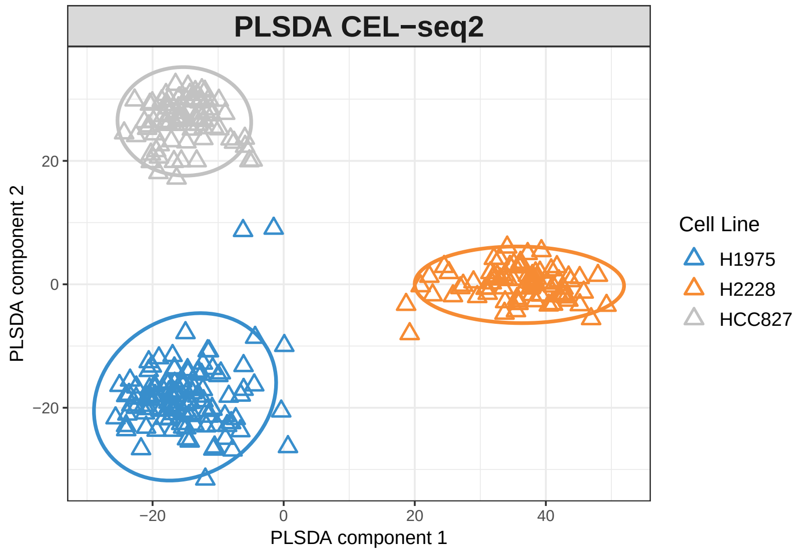


Figure 2.3: PLSDA plot for the CEL-seq2 dataset.

result of an inversion on second PLSDA component and does not imply disparity per se.

2.5.2 Sparse PLSDA (sPLSDA) for each protocol individually

For consistency and better comparison, we will use same number of optimum components as MINT for the sparse PLS-DA on each protocol:

```r
## run sparse PLSDA on individual studies with MINT tuned parameters
keepX = c(35,10)
splsda.10x.res = splsda( X =normalised.10x, Y = Y.10x, ncomp = 2,
                              keepX = keepX)
splsda.cel.res = splsda( X =normalised.cel, Y = Y.cel, ncomp = 2,
                              keepX = keepX)
splsda.drop.res = splsda( X =normalised.drop, Y = Y.drop, ncomp = 2,
                              keepX = keepX)
```

And visualise the output to see how the clusters change with variable selection:

```r
## splsda plots with tuned number of variables for each sPLSDA component
## 10X
plotIndiv(splsda.10x.res, group = Y.10x,
          legend  = T, title     = 'sPLSDA - 10X',
          ellipse = F,legend.title = 'Cell Line',
          pch=1,
          X.label = 'sPLSDA component 1',
          Y.label = 'sPLSDA component 2')
```

```r
## CEL-seq2
plotIndiv(splsda.cel.res, group = Y.cel,
          legend  = T, title     = 'sPLSDA - CEL-seq2',
          ellipse = F,legend.title = 'Cell Line',
          pch=2,
          X.label = 'sPLSDA component 1',
          Y.label = 'sPLSDA component 2')
```

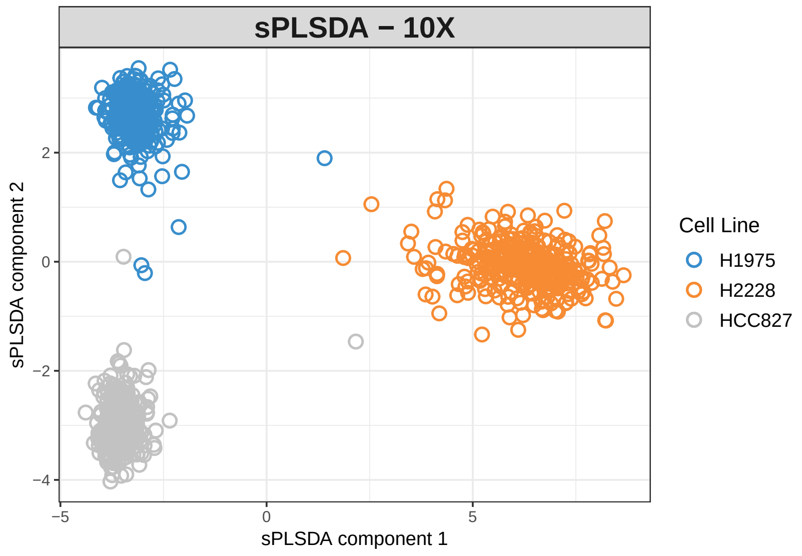


Figure 2.4: The sPLSDA plots of the 10X dataset.

The variable selection has reduced the noise and refined the clusters compared to the non-sparse model.

```r
## The signature genes from each sPLS-DA study
Chromium.10X.vars = unique(c(selectVar(splsda.10x.res, comp=1)$name,
                             selectVar(splsda.10x.res, comp=2)$name))
Cel.seq.vars =      unique(c(selectVar(splsda.cel.res, comp=1)$name,
                             selectVar(splsda.cel.res, comp=2)$name))
Drop.seq.vars =     unique(c(selectVar(splsda.drop.res, comp=1)$name,
                             selectVar(splsda.drop.res, comp=2)$name))
```

```r
## create a venn diagram from signatures
vennProtocols <- venn.diagram(
    x = list(
        Chr.10X= Chromium.10X.vars ,
        Cel.seq= Cel.seq.vars,
        Drop.seq = Drop.seq.vars),
    filename = NULL,
    cex=1.5, cat.cex=1.5,
    fill = c('green', 'darkblue',  'yellow')
    )
png(filename = 'figures/vennProtocols.png')
grid.draw(vennProtocols)
dev.off()
```

The signatures from the 3 studies share 25 common genes.

2.6 compare with MINT-combined

We can either reproduce the tuned sparse MINT object or load it from local directory, if saved:

```r
if (io$recalc){
  data.combined = t( ## transpose of all 3 datasets combined
    data.frame(
      ## the genes from each protocol that match list.intersect
```

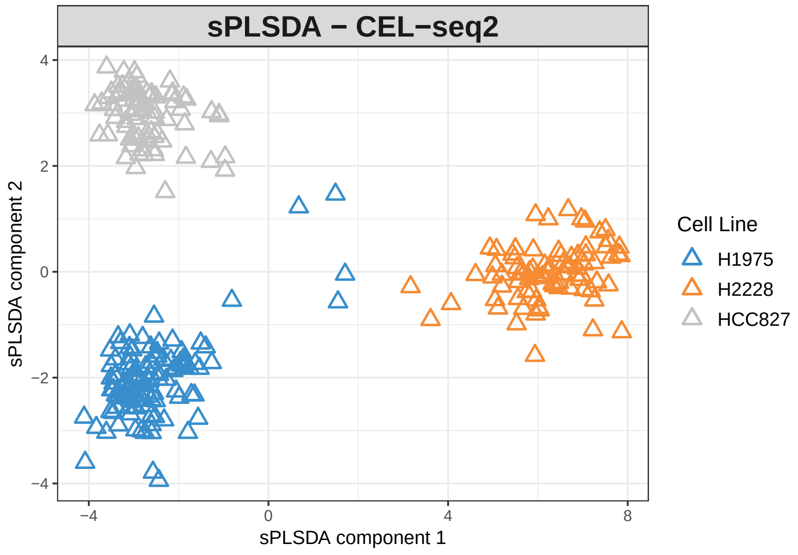


Figure 2.5: The sPLSDA plots of the CEL-seq2 dataset.

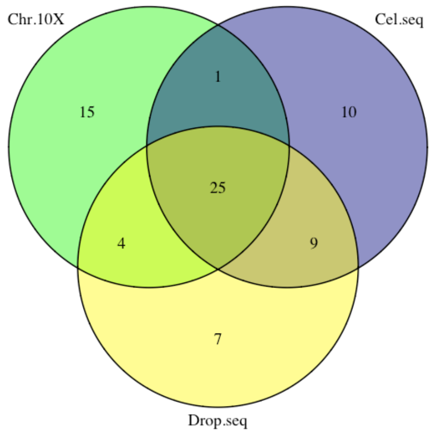


Figure 2.6: The venn diagram of common sPLSDA signatures among individual datasets.

```
    logcounts(sc10x.norm)[list.intersect,],
    logcounts(sccel.norm)[list.intersect,],
    logcounts(scdrop.norm)[list.intersect,] ))

## create a factor variable of cell lines
## must be in the same order as the data combination
cell.line = as.factor(c(sce_sc_10x_qc$cell_line,
                         sce_sc_CELseq2_qc$cell_line,
                         sce_sc_Dropseq_qc$cell_line))
## name the factor variable with the cell ID
names(cell.line) = rownames(data.combined)

## produce a character vector of batch names
## must be in the same order as the data combination
study = as.factor(
    c(rep('10X',      ncol(logcounts(sc10x.norm))),
      rep('CEL-seq2',  ncol(logcounts(sccel.norm))),
      rep('Drop-seq', ncol(logcounts(scdrop.norm))) ))
  ## name it with corresponding cell IDs
  names(study) = rownames(data.combined)

  ## run sparse mint using optimum parameters:
  mint.splsda.tuned.res = mint.splsda( X =data.combined, Y = Y,
                                study = study, ncomp = 2,
                                keepX = c(35,10)) ## change for your own dataset
} else { ## if already saved
  load(file.path('../output/mint.splsda.tuned.res.RData'))
}
```

2.6.1 loading plots

The consistency of the selected variables across individual and the MINT combined studies can be evaluated by plotting the loadings for each study using the *plotLoadings* function, which produces the barplot of variable loadings for each component.

The inputs to *plotLoadings* depends on the analysis object (mint.*s*plsda, mint.pls, etc.) and consists of the plot object plus:

- **contrib:** Whether to show the class in which the expression of the features is maximum or minimum. One of *'min'* or *'max'*.
- **method:** The criterion to assess the contribution. One of *'mean'* or *'median'* (recommended for skewed data).
- **study:** The studies to be plotted (for combined data). *all.partial* or *global* for all.

For a complete list of the arguments for any object please refer to the documentation.

We plot the loadings for both components for maximum contribution. Colours indicate the class (cell line) in which the gene is positively/negatively expressed:

```
## Loading Plots
## 10X
plotLoadings(splsda.10x.res, contrib='max', method = 'mean', comp=1,
             study='all.partial', legend=F, title=NULL,
             subtitle = '10X')
```

```
## CEL-seq2 - Comp. 1
plotLoadings(splsda.cel.res, contrib='max', method = 'mean', comp=1,
             study='all.partial', legend=F, title=NULL,
             subtitle = 'CEL-seq2')
```

```
## Drop-seq - Comp. 1
plotLoadings(splsda.drop.res, contrib='max', method = 'mean', comp=1,
             study='all.partial', legend=F, title=NULL,
             subtitle = 'Drop-seq')
```

These genes are differentiating cells along the first sPLSDA component. Majority of signature genes are postively expressed in H2228 cells (orange). There is not a clear consensus among individual datasets in terms of the signature genes and their weight (loading).

```
## MINT - Comp. 1
plotLoadings(mint.splsda.tuned.res, contrib='max', method = 'mean', comp=1,
             study='all.partial', legend=F, title=NULL,
```

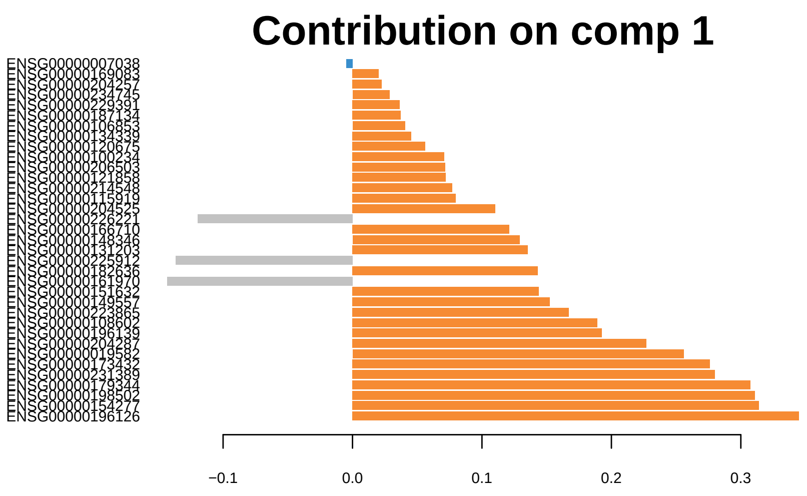


Figure 2.7: The loading plots for the first component of sPLS-DA on 10X data. The orange colour corresponds to H2228 cells, and the grey colour belongs to HCC827 cell line.

```
             subtitle = c('10X', 'CEL-seq2', 'Drop-seq') )
```

MINT has produced signature that is consistent across studies. Similarly, we can plot the loadings on the second component variables.

```
## MINT - Comp. 2
plotLoadings(mint.splsda.tuned.res, contrib='max', method = 'mean', comp=2,
             study='all.partial', legend=F, title=NULL,
             subtitle = c('10X', 'CEL-seq2', 'Drop-seq') )
```

2.6.2 signature comparison

The *selectVar* function in *mixOmics* outputs the key predictors on each component along with their loadings. We can create a set of Venn diagrams to visualise the overlap of the signature found using PLS-DA in individual datasets and the combined dataset. For all datasets, we take all the variables on both components as the signature.

```
## MINT signature
MINT.Combined.vars = unique(c(selectVar(mint.splsda.tuned.res, comp=1)$name,
                              selectVar(mint.splsda.tuned.res, comp=2)$name))
```

```
## create venn diagram
vennMINT <- venn.diagram(
    x = list(
        Chr.10X= Chromium.10X.vars ,
        Cel.seq= Cel.seq.vars,
        Drop.seq = Drop.seq.vars,
        MINT.Combined=MINT.Combined.vars),
    filename = NULL,
    cex=1.5, cat.cex=1.5,
    fill = c('green', 'darkblue',  'yellow', 'red')
```

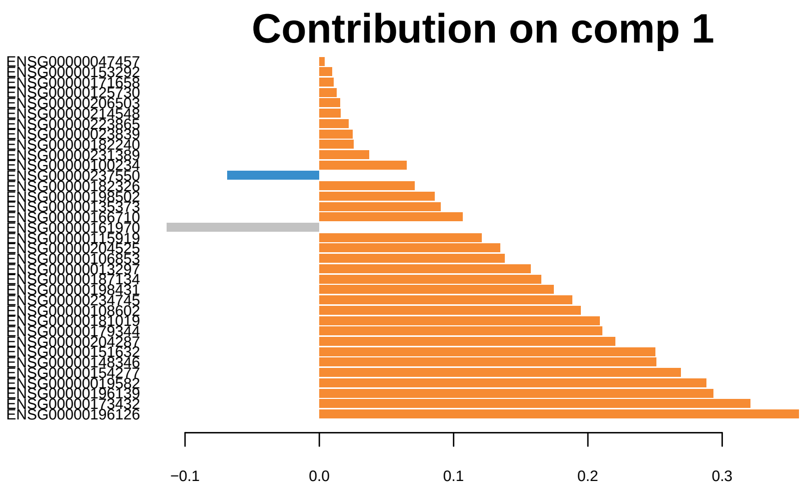


Figure 2.8: The loading plots for the first component of sPLS-DA on CEL-seq2 data.

```
    )
png(filename = 'figures/vennMINT.png')
grid.draw(vennMINT)
dev.off()
```

```
## the signature genes identified by all studies
common.sig = Reduce(intersect, list(MINT.Combined.vars, Chromium.10X.vars, Cel.seq.vars, Drop.seq.vars))
```

MINT has successfully detected the core 25 signature genes shared by all individual PLSDA models, plus 10 genes identied in 2 out of 3 studies.

```
## the 10 signature genes with highest loadings
common.sig[1:10]
```

```
##  [1] "ENSG00000196126" "ENSG00000154277" "ENSG00000173432"
##  [4] "ENSG00000019582" "ENSG00000204287" "ENSG00000198502"
##  [7] "ENSG00000231389" "ENSG00000196139" "ENSG00000179344"
## [10] "ENSG00000108602"
```

2.6.3 variable plots

Using *plotVar* function, it is possible to display the selected variables on a correlation circle plot to find the correlation between gene expressions:

```
## correlation circle plot
plotVar(mint.splsda.tuned.res, cex = 3)
```

The figure above shows that majority of the signature genes are positively expressed in component 1.

we can assess the expression profiles for the genes on the extremes of both components:

```
## show genes on extreme sides

## component 1 - most positively and negatively expressed between cell lines
```

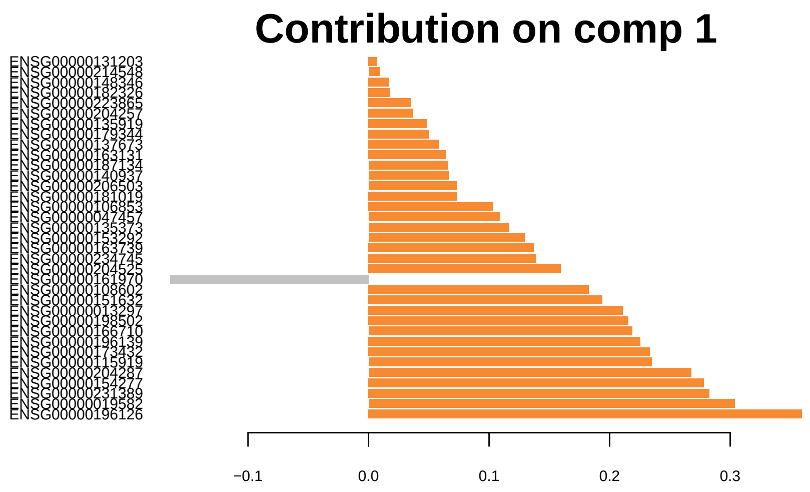


Figure 2.9: The loading plots for the first component of sPLS-DA on Drop-seq data.

```r
var.c1 = selectVar(mint.splsda.tuned.res, comp=1)$value
positive.gene.c1 = rownames(var.c1)[which.max(var.c1$value.var)]
negative.gene.c1 = rownames(var.c1)[which.min(var.c1$value.var)]

## component 2 - most positively and negatively expressed between cell lines
var.c2 = selectVar(mint.splsda.tuned.res, comp=2)$value
positive.gene.c2 = rownames(var.c2)[which.max(var.c2$value.var)]
negative.gene.c2 = rownames(var.c2)[which.min(var.c2$value.var)]
```

```r
## a function to create violin + box plots for this specific dataset
violinPlot = function(mint.object, gene){
  cols = c("H2228" = "orange", "H1975" = "dodgerblue3", "HCC827" = "grey")
  ggplot() +
    geom_boxplot(aes(mint.object$Y,  mint.object$X[,gene],
                     fill= mint.object$Y), alpha=1)+
    geom_violin(aes(mint.object$Y, mint.object$X[,gene],
                     fill= mint.object$Y), alpha=0.7) +
    labs(x = "Cell Line", y="Standardised Expression Value" )+
    guides(fill=guide_legend(title="Cell Line") ) +
    scale_fill_manual(values=cols )
}
```

```r
## violin + box plots for the most positively expressed gene on component 1
violinPlot(mint.splsda.tuned.res, positive.gene.c1)
```

```r
## violin + box plots for the most negatively expressed gene on component 1
violinPlot(mint.splsda.tuned.res, negative.gene.c1)
```

As expected, the expression profile of the above genes for H2228 cell line has opposite correlation to the other two. We can also evaluate the genes of the second sPLS-DA component:

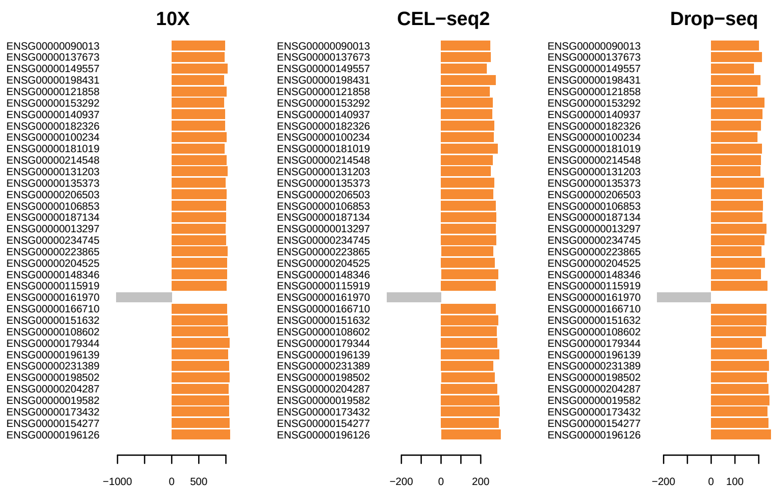


Figure 2.10: The loading plots for the first component of the Sparse MINT on the combined dataset.

```r
## violin + box plots for the most positively expressed gene on component 2
violinPlot(mint.splsda.tuned.res, positive.gene.c2)
```

```r
## violin + box plots for the most negatively expressed gene on component 2
violinPlot(mint.splsda.tuned.res, negative.gene.c2)
```

The expression profiles show that the selected gene tends to be negatively expressed in HCC827 cell line and positively expressed in H1975 cell line.

A Clustered Image Map including the final signature can be plotted. The argument *comp* can be also be specified to highlight only the variables selected on specific components.

```r
## hierarchical clustering
cim(mint.splsda.tuned.res, comp = c(1,2), margins=c(10,5),
    row.sideColors = color.mixo(as.numeric(mint.splsda.tuned.res$Y)), row.names = F,
    title = 'MINT sPLS-DA', save='png', name.save = 'heatmap')
```

The heatmap shows that the cells from each cell line tend to group together based on their gene expression profile. It can be reiterated that the cells from the H2228 cell line show positive expression in majority of signature genes.

2.7 signature comparison with pseudo-bulk assay

2.7.1 cell mixtures data

The experimental design for CellBench data also included a cell mixture design for a pseudo-trajectory analysis. Briefly, in this study 9 cells were pooled from different cell types in different proportions and then reduced to 1/9th to mimic single cells and were sequenced using CEL-seq2 protocol. A mixture of 90 cells in equal amounts was used as the control group for a Differential Expression (DE) analysis of the pure cell lines at the extreme of trajectories. The results available on CellBench repository.

We will take DE genes from the cell mixture study as reference for comparison. The DE genes are available on CellBench respository.

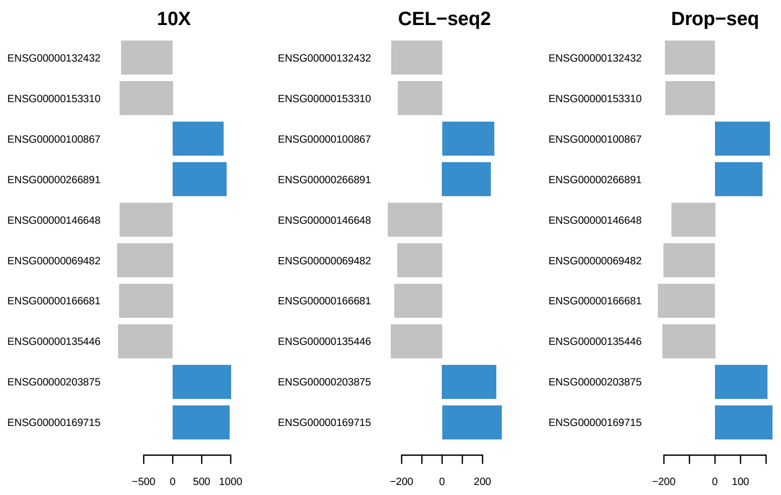


Figure 2.11: The loading plots for the second component of the Sparse MINT on the combined dataset.

```
if (isFALSE(io$local.input)){
  ## load from GitHub
  DE_URL <- 'https://tinyurl.com/DEtable-90cells'
  load(url(DE_URL))
} else {
  load(file.path(io$local.input, 'DEtable_90cells.RData'))
  }
```

2.7.2 DE genes vs MINT signature

We now compare the unique signature genes from MINT sPLS-DA analysis to the ones from cell mixture DE analysis:

```
## keep the genes present in the sPLS-DA analysis
HCC827_DEtable = HCC827_DEtable[row.names(HCC827_DEtable) %in% list.intersect,]
H2228_DEtable = H2228_DEtable[row.names(H2228_DEtable) %in% list.intersect,]
H1975_DEtable = H1975_DEtable[row.names(H1975_DEtable) %in% list.intersect,]

## create a column of genes for ease of merging
HCC827_DE = rownames_to_column(HCC827_DEtable, 'gene')
H2228_DE = rownames_to_column(H2228_DEtable, 'gene')
H1975_DE = rownames_to_column(H1975_DEtable, 'gene')

## create a combined DEtable from the 3 cell lines
DEtable = rbind(HCC827_DE,H2228_DE, H1975_DE)
## order by gene name and FDR (increasing)
DEtable = DEtable[order(DEtable[,'gene'],DEtable[,'FDR']),]
## keep the ones with FDR<0.05
DEtable = DEtable[DEtable$FDR<0.05,]
## remove duplicate gene names
DEtable = DEtable[!duplicated(DEtable$gene),]
## overlap with MINT
DE.MINT = DEtable[(DEtable$gene %in% MINT.Combined.vars),]
```

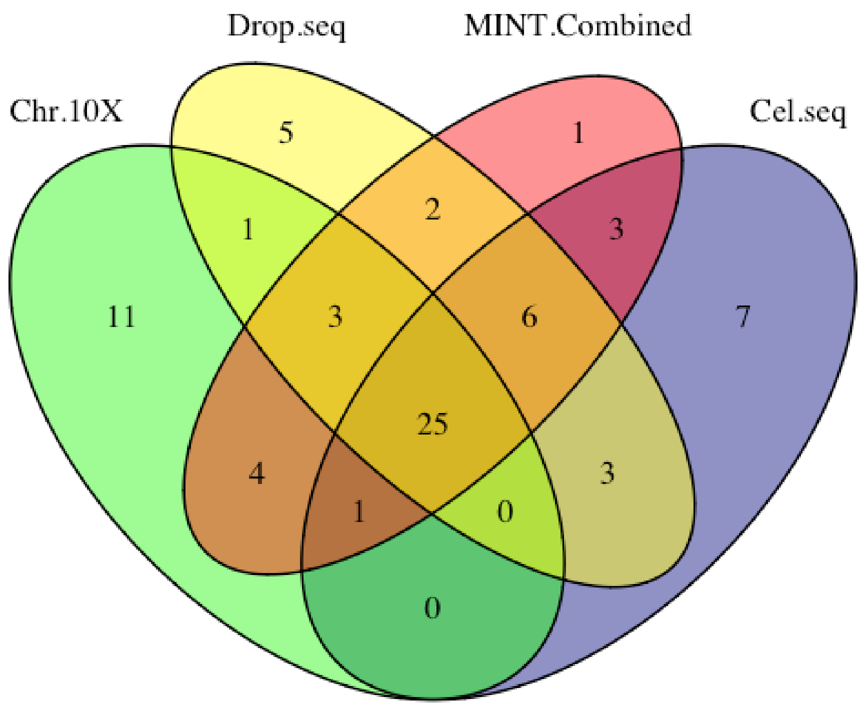


Figure 2.12: The venn diagram of sPLSDA signatures among individual datasets and the MINT combined dataset.

```
## sort in increasing FDR order
DE.MINT = DE.MINT[order(DE.MINT$FDR),]
## number of MINT signature genes that are differentially expressed
dim(DE.MINT)[1]
```

```
## [1] 45
```

```
## geometric mean of the FDR of signature
exp(mean(log(DE.MINT$FDR)))
```

```
## [1] 1.548651e-18
```

All of signature genes identified by MINT are differentially expressed at $FDR<0.05$ with geometric mean FDR of 1.54×10^{-18}.

2.8 session information

```
## session information to build this vignette
sessionInfo()
```

```
## R version 3.5.0 (2018-04-23)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS  10.14.1
```

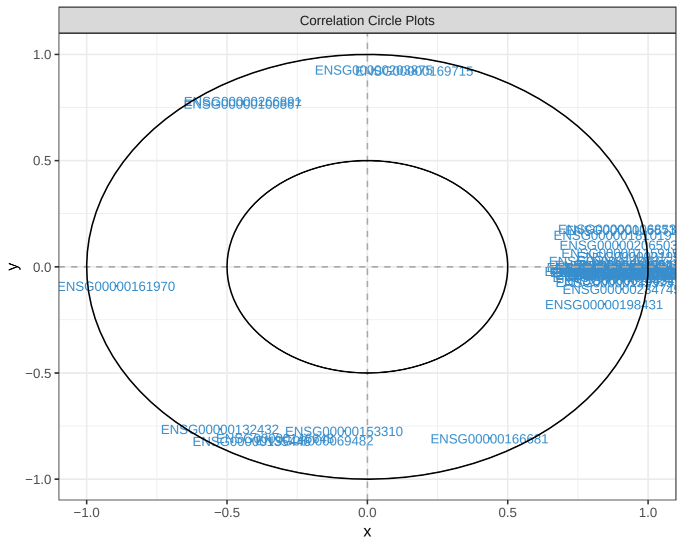


Figure 2.13: The variable plot highlighting the contribution of each selected variable to each component and their correlation.

```
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
##  [1] grid      parallel  stats4    stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] tibble_1.4.2                VennDiagram_1.6.20
##  [3] futile.logger_1.4.3         scran_1.9.39
##  [5] mixOmics_6.7.0              ggplot2_3.1.0
##  [7] lattice_0.20-38             MASS_7.3-51.1
##  [9] SingleCellExperiment_1.3.12 SummarizedExperiment_1.11.6
## [11] DelayedArray_0.7.49         BiocParallel_1.15.15
## [13] matrixStats_0.54.0          Biobase_2.41.2
## [15] GenomicRanges_1.33.14       GenomeInfoDb_1.17.4
## [17] IRanges_2.15.19             S4Vectors_0.19.24
## [19] BiocGenerics_0.27.1         knitr_1.20
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-6              RColorBrewer_1.1-2
##  [3] rprojroot_1.3-2           dynamicTreeCut_1.63-1
##  [5] tools_3.5.0               backports_1.1.2
##  [7] R6_2.3.0                  HDF5Array_1.9.19
##  [9] vipor_0.4.5               lazyeval_0.2.1
## [11] colorspace_1.3-2          withr_2.1.2
## [13] tidyselect_0.2.5          gridExtra_2.3
```

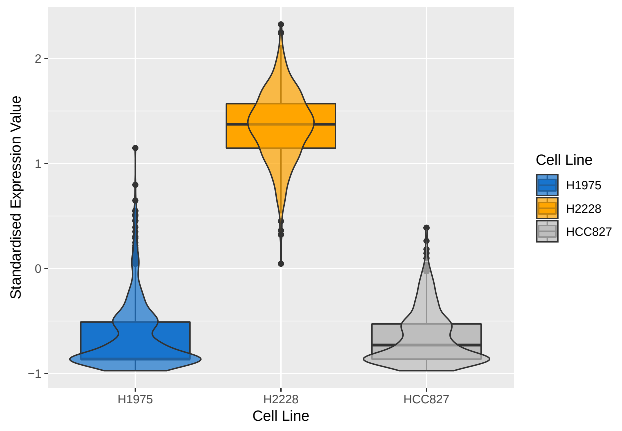


Figure 2.14: Violin plots of expression profile of the most positively expressed gene on component 1 in different cell lines

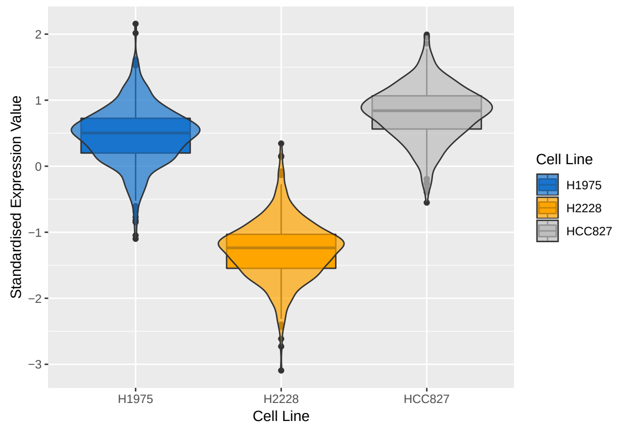


Figure 2.15: Violin plots of expression profile of the most negatively expressed gene on component 1 in different cell lines

```
## [15] compiler_3.5.0            BiocNeighbors_0.99.22
## [17] formatR_1.5               labeling_0.3
## [19] bookdown_0.7              scales_1.0.0
## [21] stringr_1.3.1             digest_0.6.18
## [23] rmarkdown_1.10            XVector_0.21.4
## [25] scater_1.9.24             pkgconfig_2.0.2
## [27] htmltools_0.3.6           limma_3.37.10
## [29] rlang_0.3.0.1             DelayedMatrixStats_1.3.11
## [31] bindr_0.1.1               dplyr_0.7.8
## [33] RCurl_1.95-4.11           magrittr_1.5
## [35] GenomeInfoDbData_1.2.0    Matrix_1.2-15
## [37] Rcpp_1.0.0                ggbeeswarm_0.6.0
## [39] munsell_0.5.0             Rhdf5lib_1.3.3
## [41] viridis_0.5.1             stringi_1.2.4
## [43] yaml_2.2.0                edgeR_3.23.7
## [45] zlibbioc_1.27.0           rhdf5_2.25.11
## [47] plyr_1.8.4                crayon_1.3.4
## [49] locfit_1.5-9.1            pillar_1.3.0
```

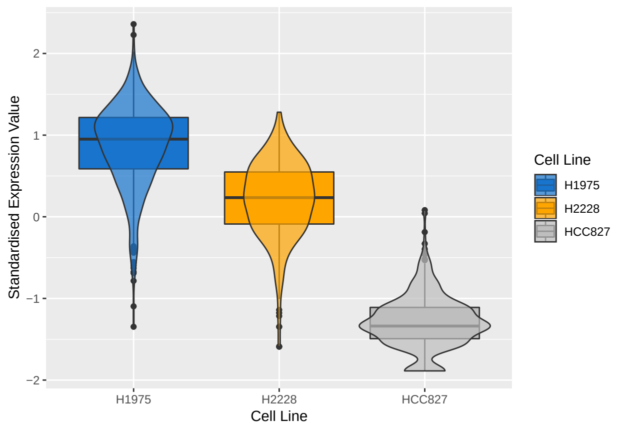


Figure 2.16: Violin plots of expression profile of the most positively expressed gene on component 2 in different cell lines

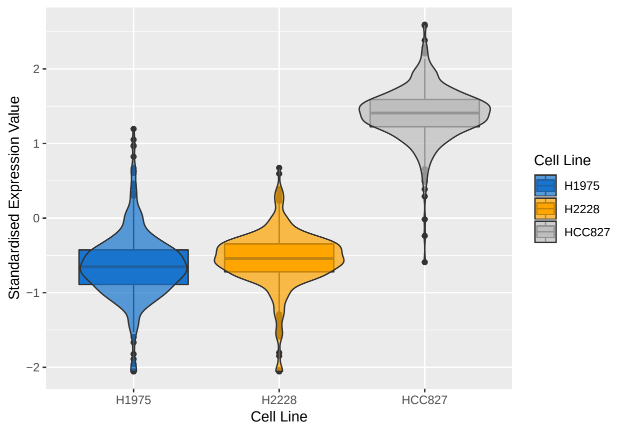


Figure 2.17: Violin plots of expression profile of the most negatively expressed gene on component 2 in different cell lines

```
## [51] igraph_1.2.2          corpcor_1.6.9
## [53] reshape2_1.4.3        codetools_0.2-15
## [55] futile.options_1.0.1  glue_1.3.0
## [57] evaluate_0.12         lambda.r_1.2.3
## [59] BiocManager_1.30.4    gtable_0.2.0
## [61] purrr_0.2.5           tidyr_0.8.2
## [63] assertthat_0.2.0      xfun_0.4
## [65] RSpectra_0.13-1       viridisLite_0.3.0
## [67] rARPACK_0.11-0        beeswarm_0.2.3
## [69] ellipse_0.4.1         bindrcpp_0.2.2
## [71] statmod_1.4.30
```

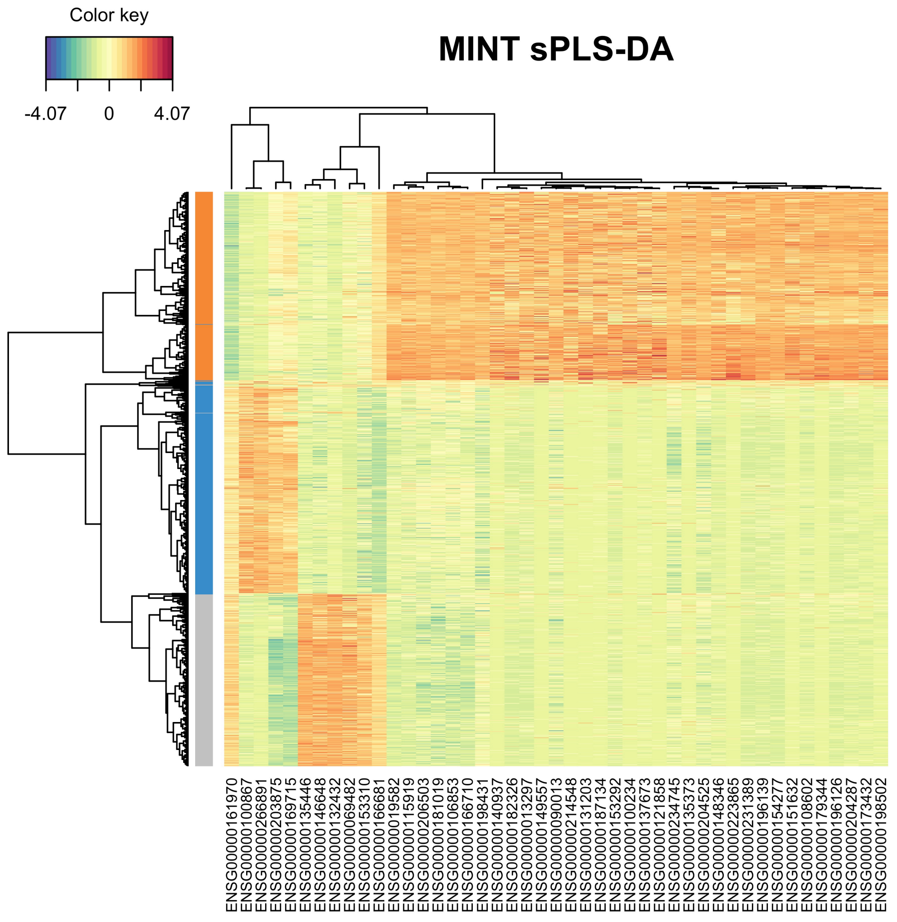


Figure 2.18: The heatmap showing the cell grouping based on the gene expression profiles.

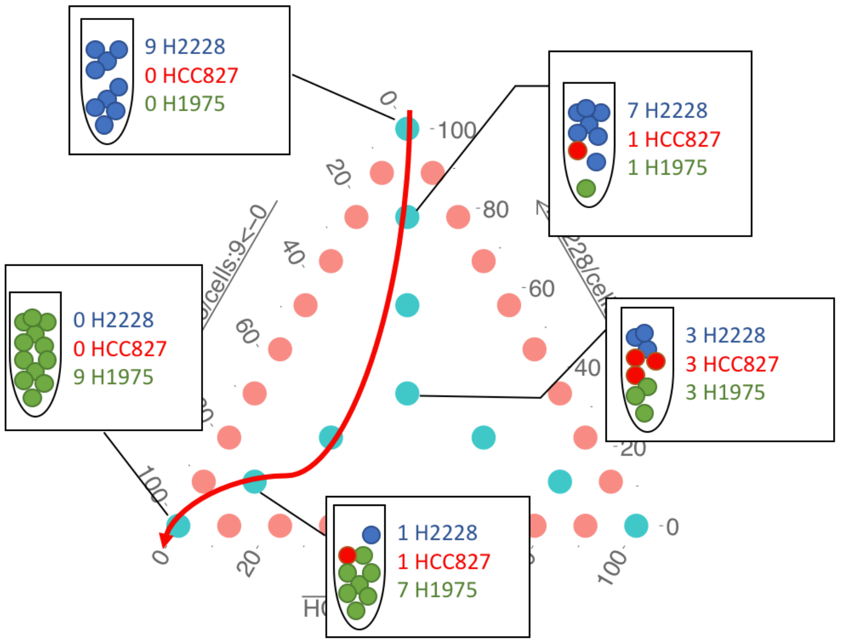


Figure 2.19: The cell mixture design. Adopted from Luyi Tian slides

Bibliography

Jolliffe, I. T. (1986). *Principal Component Analysis and Factor Analysis.* New York, NY: Springer New York, pp. 115–128.

Le Cao, K.-A., Rohart, F., Gonzalez, I., with key contributors Benoit Gautier, S. D., Bartolo, F., contributions from Pierre Monget, Coquery, J., Yao, F., and Liquet., B. (2017). *mixOmics: Omics Data Integration Project.* R package version 6.7.0.

Lun, A. T. L., Bach, K., and Marioni, J. C. (2016). *Pooling across cells to normalize single-cell RNA sequencing data with many zero counts.*

Rohart, F., Eslami, A., Matigian, N., Bougeard, S., and Cao, K.-A. L. (2017). *MINT: a multivariate integrative method to identify reproducible molecular signatures across independent experiments and platforms.*

Tian, L. and Su, S. (2018). *scPipe: pipeline for single cell RNA-seq data analysis.* R package version 1.3.8.