SelectBCM tool: Short introduction

"Madhulika Mishra, Lucas Barck, Pablo Moreno, Guillaume Heger, Yuyao Song, Janet M. Thornton, Irene Papatheodorou"

maintained by: madhulika mishra(madhulika@ebi.ac.uk)

Introduction

Integrating information from bulk transcriptomes from different experiments can potentially increase the impact of individual studies in areas such as biomedicine. Despite that essential data sources for basic and disease biology are available, adequate integration remains difficult. Batch-effect correction for both experimental and technological variations is one of the main challenges in transcriptomics studies. To handle this, various batch-correction methods(BCMs) have been developed in recent times. However, the lack of a user-friendly workflow to select the most appropriate BCM for the given set of experiments needs to be resolved for applying batch correction at a large scale e.g. for multi-cohort transcriptomic analysis. In this project, we developed an R tool named "SelectBCM Tool" to increase the reusability and reproducibility of these BCMs, in order to facilitate the application, comparison and selection of BCMs. Our tools aims to aid choosing the most suitable batch correction method for a set of bulk transcriptomic experiments.

Project overview

SelectBCM performes the following analysis:

- Fetching experiments from ExpressionAtlas based on a biological factor of interest, such as disease, species and tissue
- Removing isolated experiments
- Merging experiments in a single dataset
- detetction of batch effect in the merged dataset
- Correcting batch effects
- Evaluation of batch correction methods

The above steps have been implemented in a variety of forms including stand-alone tools, scripts or R package functions. By creating SelectBCM we aim to provide the infrastructure to compare equivalent logical steps between workflows, and to 'mix and match' those components for a custom optimal workflow. The current R package provides a flexibility to perform batch correction and evaluation of various batch-correction methods for given dataset. Finally, it provide **performance rank** of each BCMs for the given dataset.

Installation

It is advised to set up a dedicated conda environment for a new SelectBCM project. To use SelectBCM with conda, run the following commands from a bash shell:

```
conda env create -n selectBCM -c r r-base=4.2.2
conda activate selectBCM
conda install -c conda-forge r-remotes
R
```

In the R session, run the following installation commands:

```
remotes::install_github("tengfei-emory/scBatch", ref="master")
remotes::install_github("ebi-gene-expression-group/selectBCM", ref="master", build=FALSE)
```

Alternatively, we also created Docker images to run SelectBCM R package inside a Docker container on either linux/amd64 or linux/arm64 platforms. We also provide the Docker file of the SelectBCM Docker images for custom container building from scratch.

linux/amd64:

Pull container from Docker and open an interactive R session:

```
docker pull yysong123/selectbcm:amd64
docker run -it yysong123/selectbcm:amd64
```

linux/arm64:

```
docker pull yysong123/selectbcm:arm64
docker run -it yysong123/selectbcm:arm64
```

For a renv approach of version control, we provide `renv.lock` for the package dependencies. Install the specific package versions as recorded in `renv.lock` using `renv::restore()` at the local package directory.

SelectBCM can be installed via github-

```
install_github('https://github.com/ebi-gene-expression-group/selectBCM')
```

Or download the package as `zip` archive and install-

```
install.packages('selectBCM-master.zip', repos = NULL, type = 'source')
```

Overview of steps available in SelectBCM

SelectBCM package has several steps ranging from meta-experiment creation to batch-effect evaluation step. In the current wrapper, scripts are written in a user-friendly way. Short description of each step and example is given below -

loading library

Sometime loading of 'magrittr','purrr' and 'dplyr' with `SelectBCM` package is deprecated, therefore, it is recommended to load all of these together.

Recommendation: Computation of scBatch requires high memory allocation for Rcpp, therefore it is advised to increase R memory.

```
library(DESeq2)
library(SelectBCM)
library(magrittr)
library(dplyr)
library(purrr)
library(tibble)
library(gtools)
library(readr)
library(Biobase)

library(ggplot2)
library(scales)
library(ggpubr)
```

Step1: Getting data

Loading local data in R: If the data files are already on the user's local machine, they can be used directly as follows:

Steps:

- Put the data files in `.Rdata` format in a directory containing only them. (The R objects contained in those files must be either `SummarizedExperiment` or `ExpressionSet` experiment.)
- Load all the experiments in a list using the function `load_experiments`

```
experiments <- load_experiments('directory_path')
experiments<-load_experiments('./')
```

If user want to download data from Expression Atlas, they can skip the above and go directly to "Downloading data from Expression Atlas".

Downloading data from Expression Atlas: In case of data downloading from Expression Atlas, user can use the function `download_experiments_from_ExpressionAtlas` in this way :

```
experiments <- download_experiments_from_ExpressionAtlas('E-MTAB-8549','E-MTAB-5060')
```

*** will download RNAseq experiments from expressionatlas.

Recommendation: Microarray input data should be preprocessed- appropriately back-ground corrected without any normalization step, probe-to-gene level mapped and log-transformed. It is also recommended to remove low-expressed genes from data if possible. For bulk-RNaseq data, it should be a count matrix.

Output from step 1: This downloads the experiments in a new directory called **"experiments"** in user's working directory and loads all the experiments in R within a list, using `load_experiments` function. After having loaded the experiments, user will get a list of either `SummarizedExperiment` or list of `ExpressionSet` objects.

Caution: Avoid mixing experiments of `SummarizedExperiment` with `ExpressionSet`. Experiments can only belong from any one of the classes only. All the selected experiments should have same gene id format.

Step2: Removing the isolated experiments

To correct the `batch effect`, one needs to take the biological characteristics of the samples into account (`gender` in our example). If no sample of an experiment shares biological characteristics with samples from other batches, it is not possible to correct batch effect with these batches because one cannot distinguish the biological difference from the artifact. The function `remove_isolated_experiments` removes the isolated experiments and plots graphs of intersections between the experiments before and after removal.

```
experiments %<>% remove_isolated_experiments('sex')
```

WARNING: this function only removes the isolated experiment according to the supplied key. Although it is still possible that two or more unconnected groups of experiments remain, within which the experiments are connected. In this case, batch effect correction is not possible and one will have to choose a group of experiments manually.

The two following plots are displayed by the function. The first one shows the graph of intersections of all the experiments before the removal of isolated ones. The second shows the same graph after their removal (Figure2).

Step3: Merging experiments in a single dataset

The function `merge_experiments` merges all the experiments in the list in a single `SummarizedExperiment` or `ExpressionSet` object and doesn't perform any correction. This function has two additional arguments `log` and `filter` (respectively set to `TRUE` and `FALSE` by default).

- For RNAseq experiments: users have to call `merge_experiment.SummarizedExperiment()` function.
- For microarray experiment, users have to call `merge_experiment.ExpressionSet()` function.
- The `log` argument determines whether to perform log transformation on the data (recommended for bulk RNAseq).

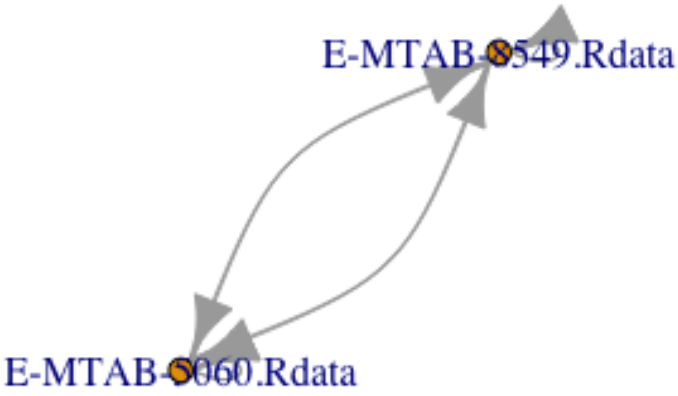

Figure 1: Figure2. Graph showing connected set of experiments.

- The `filter` argument determines whether to filter genes for which all the samples of a batch have zero-counts. Set it to `TRUE` if user have issues in running ComBat at the next step.

```
experiments %<>% merge_experiment.SummarizedExperiment()
```

`experiments` is now an only `ExpressionSet` object containing the information about batches both in its `@metadata` and `@colData` slots :

Caution: Sometimes during merging experiment, phenodata (SDRF) file gets corrupted, hence, it is advised to always check meta-data before proceeding further.

Step4: Correcting batch effect

The function `batch_correction` performs various methods of batch-correction on a given merged dataset and output batch-corrected data as a list.

Short detail of methods implemented in `batch_correction` function are given below-

- For Microarray, `batch_correction` function has following batch correction methods-
  - **limma**- default limma setting, more detail about the method can be learnt from Limma documentation.
  - **GFS**- Gene Fuzzy Score, more detail about the method can be learnt from publication.
  - **Robust quantile normalization**- quantile normalisation method from PreprocessCore package. More information can be obtained from their documentation.
  - **ComBat**- implemented from SVA package. More information can be obtained from their documentation. In the current `batch_correction` method, there are two versions of ComBat- 1) **ComBat1**- for parametric adjustment and, 2) **ComBat2** - for non-parametric adjustment, mean-only version.
  - **Q_ComBat** - is Quantile+ parametric adjustment of ComBat.
  - **MNN** - default mnnCorrect function from batchelor package. More information can be obtained from their documentation.
  - **naiveRandRUV_HK** - default naiveRandRUV using Human HK genes from RUVnormalize package. More information can be obtained from their documentation. The Human Housekeeping gene list is from this publication. No. of confounders were estimated using both "leek" and "be" method and then least no. of estimated confounders were used as input for all variants of **naiveRandRUV** method.
  - **Q_naiveRandRUV_HK** - is Qunatile+ naiveRandRUV_HK
  - **naiveRandRUV_empi.controls** - default naiveRandRUV using data-empirically derived HK genes from RUVnormalize package.
  - **Q_naiveRandRUV_empi.controls** -is Qunatile+ naiveRandRUV_empi.controls.
- For bulk-RNAseq, `batch_correction` function has the following batch correction methods-
  - **limma**- default limma setting, more detail about the method can be learnt from Limma documentation.
  - **ComBat**- implemented from SVA package. More information can be obtained from their documentation.
  - **ComBat_seq** - RNAseq version of ComBat, now implemented in `SVA` package. More information can be obtained from this publication.
  - **MNN** - default mnnCorrect function from batchelor package. More information can be obtained from their documentation.
  - **RUVs** - default RUVs function from RUVSeq package.Here, k=1 is used. More information can be obtained from their documentation.
  - **scBatch** - batch correction method implemented in scBatch package. More information can be obtained from their publication.

This function has following arguments-

- The `experiment` argument is the input merged dataset.

- The `model` argument is a R formula mentioning the biological factor to take into account during correction.
- The `batch` argument is a meta-data column which has information about batch-label.
- The `k` argument is a no. of confounders for RUV methods and denotes the "number of factors of unwanted variation to remove". If k is not provided it will then be estimated from input data.
- The `filter` argument specifies the gene label for the given dataset. It Should be one of the following string- 'symbol', 'ensembl_gene_id' or 'entrezgene_id' depending on the gene label for the given dataset.

`batch effect` present in the example study can be detected as follows:

```
dds_final <- DESeqDataSetFromMatrix(countData = assay(experiments  ),
                                    colData = colData(experiments), design = ~sex)

####check batch effect in the initial dataset:

vst3 <- varianceStabilizingTransformation(dds_final)

plotPCA(vst3,"batch")

dds_final <- estimateSizeFactors(dds_final)
se <- SummarizedExperiment(log2(counts(dds_final, normalized=TRUE) + 1),
                           colData=colData(experiments))
# the call to DESeqTransform() is needed to trigger our plotPCA method.
plotPCA( DESeqTransform( se ),intgroup =c("batch") )
```

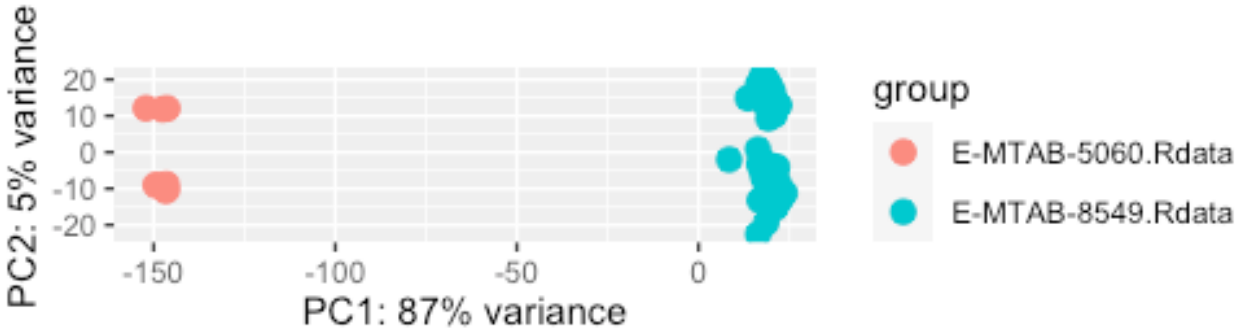

Figure 2: Figure3.

Batch_correction using SelectBCM tool:

```
result.1 <- batch_correction(dds_final,model=~sex, batch = "batch")
result.1$data.uncorrected <- se
result.1$data.uncorrected1 <- vst3
```

Output: is the list of batch-corrected data:

```
summary(result.1)

                    Length Class                  Mode
data.limma          31686  SummarizedExperiment S4
data.ComBat         31686  SummarizedExperiment S4
```

```
data.ComBat_Seq_full 31686  SummarizedExperiment S4
data.ComBat_seq_null 31686  SummarizedExperiment S4
data.MNN             31686  SummarizedExperiment S4
data.RUVs            31686  SummarizedExperiment S4
data.scBatch         31686  SummarizedExperiment S4
data.uncorrected     31686  SummarizedExperiment S4
data.uncorrected1    31686  DESeqTransform       S4
```

Step5: Assessment of batch-correction methods

The `batch_evaluation` allows users to performs various evaluations on batch-corrected data and output performance list of each method. Since there are no. of ways batch-correction can be evaluated and each method has some limitation, we have used a cocktail of methods to perform analysis. This function has both PCA-inspired as well as biology inspired qualitative assessment protocol for batch-correction.

- For RNAseq experiments: users have to call `batch_evaluation.RNAseq()` function.
- For microarray experiment: users have to call `batch_evaluation.microarray()` function.

Short detail of methods implemented in `batch_evaluation` function is given below-

- **pvca**- A function for principal variance component analysis. The function is written based on the 'pvcaBatchAssess' function of the PVCA R package and slightly changed to make it more efficient and flexible for both microarray and bulk RNAseq gene-expression data. From https://github.com/dleelab/pvca.
- **silhouette**- Determine `batch effect` using the silhouette coefficient (adopted from scone) with default setting (nPcs=3). Taken from `kBET` package.
- **pcRegression**- Determines `batch effect` by a linear model fit of principal components and a batch (categorical) variable with a default setting (n_top=20). Taken from `kBET` package.
- **entropy**- Determines the batch effect by computing the entropy of mixing. It is a parameter to quantify the extent of the intermingling of cells from different batches. Taken from `MNN` package.For calculation, first two Pcs are used as input. Since,depending on no. of samples in merged dataset, it is important to choose N1 and N2 for Batchentropy calculation, we have put both arguments as variables here.
- **HVG.union**- This analysis calculates fraction of conserved highly variable genes (HVG) according to Brennecke et al., 2013 publication. This calculation also accounts for perseverance of biological heterogeneity by measuring ratio of conservation of HVG genes after batch correction. This function calculates ratio of number of conserved HVG after batch correction/ Union of HVG genes among all the batches. Therefore, this function is less stringent compared to `HVG.intersection` function.

This function has following arguments-

- The `result` argument is a list of wrapped batch-corrected experiments obtained from the last step ('batch_correction').
- The `batch.factors` is a list of factors to perform `PVCA` analysis. Along with `batch` as factor, one `biological factor` which can be used to assess over-fitting should be provided. Providing more than one `biological factor` here, will create issues while plotting the results in next step.
- The `experiment` argument is the input merged dataset.
- The `N1` is the number of randomly picked samples for the BatchEntropy function.
- The `N2` is the number of nearest neighbours of the sample (from all batches) to check (for BatchEntropy function).
- The `filter` argument specifies the gene label for the given dataset. It Should be one of following string- 'symbol', 'ensembl_gene_id' or 'entrezgene_id' depending on the gene label for the given dataset.

Assessment of batch-corrected data can be performed as follows:

```
assess.RNAseq <- batch_evaluation.RNAseq(result.1, batch.factors=c("batch","sex"),
                                         se,5,5,'ensembl_gene_id')
```

Output: assess.RNAseq is a nested list of evaluation scores for each of the evaluation protocols.

Step6: Ranking of BCMs

Once assessment is performed, in next step, results obtained from the `batch_evaluation` step, ranking of BCM is the two step analysis-

- Generation of Diagnostic matrix
- Plot of Ranks of BCM obatined from Diagnostic matrix

Diagnostic matrix

`diagnostic_matrix.RNAseq` function performs ranking of evaluation data obtained at the previous step. Here, methods are ranked based on their performance and finally `sumRank` is the final Rank of each method for the given input dataset. Rank 1 will be the best performer method.

- For RNAseq experiments: users have to call `diagnostic_matrix.RNAseq` function.
- For microarray experiment: users have to call `diagnostic_matrix.microarray` function.

This function has only one argument-

- `diagnostic` is an evaluation list obtained from the previous step `batch_evaluation`.

```
final <- SelectBCM::diagnostic_matrix.RNAseq(assess.RNAseq)
```

Output1: A list of two data-frame- (a) raw - simple data-frame output of evaluation matrix and, (b) ranked- Ranked data-frame of evaluation matrix which has additional column `sumRank` containing final Rank of each method. Ranks are in descending performance order, i.e. the method having score 1 will be the best method.

diagnostic_plot

Using output1, selectBCM provides-

a) Diagnostic plot showing the performance of BCMs across evaluation methods, where x-axis is the evaluation protocol and y-axis is the Rank of each batch-correction method.
b) violin plot to summarise the performance of BCMs

- For RNAseq experiments: users have to call `diagnostic_plot.RNAseq` function.
- For microarray experiment: users have to call `diagnostic_plot.microarray` function.

```
dia_plot <-  SelectBCM::diagnostic_plot.RNAseq(final)

svg("Diagnostic_plot_RNAseq_example.png")

ggarrange(dia_plot$diagnostic_plot,dia_plot$diagnostic_stat_summary,
          labels = c("a.", "b."),
          legend = "right", ncol = 1, nrow = 2
)

dev.off()
```

Rank plot

A bar plot representation of the BCMs where the 1st rank represents the most appropriate BCM for the given set of experiments using sumRank score.

```
bcm_ranking(final)
```

Additional function: Detetction of batch-effects in raw merged dataset

`detect_effect` performs a subset of evaluation tests of `batch_evaluation` function and provides estimates whether the merged dataset obtained after 'merge_experiments' requires batch correction or not. Higher

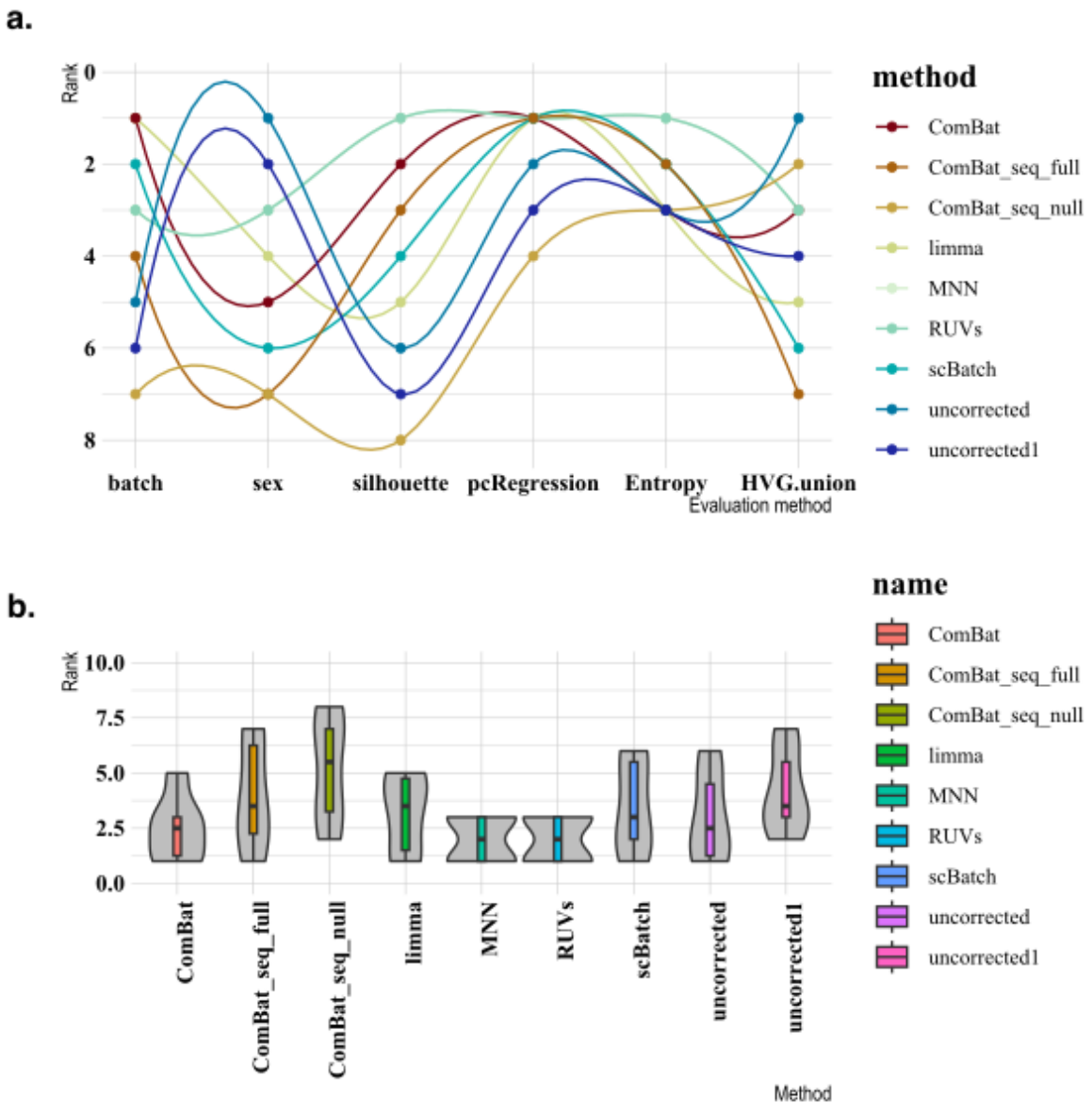

Figure 3: Diagnostic plot.

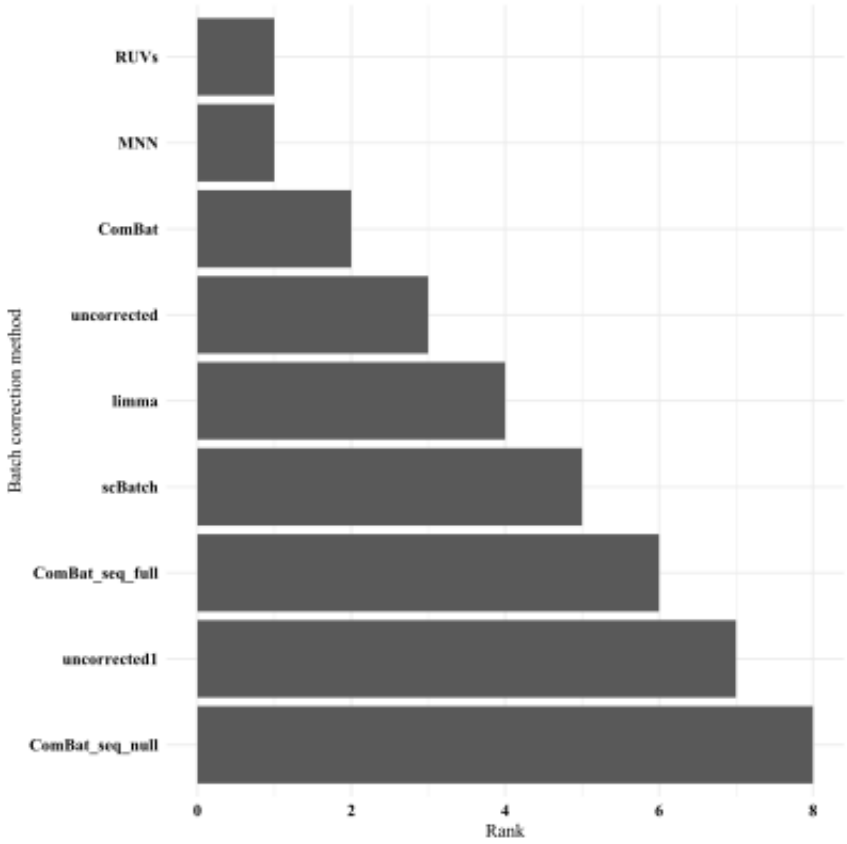

Figure 4: Diagnostic plot.

value of pvca.batch, silhouette, pcRegression and entropy is a direct indicative of batch-effects in raw merged dataset.

This function has following arguments-

- The `result` argument is a is the merged dataset obtained from `merge_experiments` step.
- The `batch.factors` is a list of factors to perform `PVCA` analysis. Along with `batch` as factor, one `biological factor` which can be used to assess over-fitting should be provided. Providing more than one `biological factor` here, will create issues while plotting the results.
- The `experiment` argument is again the same merged dataset.
- The `N1` is the number of randomly picked samples for the BatchEntropy function.
- The `N2` is the number of nearest neighbours of the sample (from all batches) to check (for BatchEntropy function).
- The `filter` argument specifies the gene label for the given dataset. It Should be one of the following string- 'symbol', 'ensembl_gene_id' or 'entrezgene_id' depending on the gene label for the given dataset.

Assessment of batch-effect on raw merged data can be performed like this:

```
batch_effect.raw <- detect_effect(experiments,experiment = experiments,
                                  batch.factors=c("batch","sex"),5,5,'ensembl_gene_id')
Result <- do.call(rbind, lapply(batch_effect.raw, as.matrix)) %>% t %>% as.data.frame()
```

Result:

```
Result

       batch sex silhouette pcRegression Entropy
raw 0.6578993   0  0.7302323    0.8403614       0
```