microbialPhenotypes: An R package that analyzes high-throughput microbial phenotype data

by Peter I-Fan Wu*, James C. Hu, Deborah A. Siegele

* Corresponding author

Abstract

Various microbial high-throughput phenotyping techniques have been vastly conducted to infer functions of genes, generating large numbers of valuable datasets whose potential in providing insights to characterize genes hasn’t been fully exploited. Therefore, computational tools that allow unbiased, systematic analysis of these data also have become vital. Here we describe a package that evaluates high-throughput microbial phenotype data by one or several sets of associated functional annotations are provided. In addition, some helper functions are provided to help clean high-throughput microbial phenotype data.

Introduction

Phenotypes play important roles in characterizing the functions of genes, particularly in microbiology, where the largest number of tests could be done much easier (Tohsato & Mori, 2008). With rising technologies (Kritikos et al., 2017; Nichols et al., 2011; Wetmore et al., 2015), high-throughput experimental approaches that measure large number of phenotypes under various conditions have flourished with complementary statistical methods in querying the behavior of gene products (Collins, Schuldiner, Krogan, & Weissman, 2006; Nichols et al., 2011; Price et al., 2018; Rishi et al., 2020). Although there are already many computational approaches written as R packages to analyze phenotype data (Deng, Gao, Wang, & Guo, 2015; Vaas et al., 2013; Vuckovic, Gasparini, Soranzo, & Iotchkova, 2015) (Vehkala, Shubin, Connor, Thomson, & Corander, 2015), software that quickly tests the potential of such high-throughput data in interpreting gene functions is lacking. Here we have wrapped the analytical pipeline of evaluating the phenotype data using annotation sets into a package named microbialPhenotypes. In this package, we provide 8 functions that are specific in dealing with high-throughput microbial phenotype data, as well as 10 functions that are meant to be more supplementary. A high-throughput E. coli phenotype dataset (Nichols et al., 2011) is used for the examples provided below. We note that there is room for significant improvement when the analytical pipeline is improved. Despite that the functions provided here started from a perspective of gaining biological insights for microbial phenotype data, the potential of using them as a tool for more general purposes should not be limited. If there are data from other
research domains that has a similar structure to the example described here, the utility of this package can be much more extended. For example, data from animal cells (Alonezi et al., 2016, 2017). In addition, knowing that there are resources for complicated machine learning algorithms to do classification of functions, our work here does not aim to improve those methods. Rather, it tries to quickly assess the usefulness of the newly generated high-throughput phenotype data before implementing more advanced classification schema.

**Installation and functions**

The [microPhenotypes](https://github.com/peterwu19881230/microbialPhenotypes) package can be downloaded from GitHub: https://github.com/peterwu19881230/microbialPhenotypes. To load the functions, source the .R files under R/.

**Input data**

This package assumes the input phenotypic profiles be in flat files (e.g., csv or tsv files) in a matrix format where each row represents a mutant strain and each column is a condition where the phenotypes of the mutant strains are assayed. The values in the columns represent the phenotypes.

Users can read their data into R using either the read.csv() or read.table() function with the appropriate arguments based on the format of the file.

For example:

```r
> phenotype_data <- read.csv(file="my_phenotype_profile.csv", header = TRUE)
> head(phenotype_data)

|           | STREPTOMYCIN.0.05 | SUCCINATE | SULFAMONOMETHOXINE.100 | TAUROCHOLATE.0.1 | TAUROCHOLATE.0.5 |
|-----------|-------------------|-----------|-------------------------|------------------|------------------|
| ECK3997-purD | -1.48             | -12.30    | 3.33                    | -13.46           | -13.79           |
| ECK0516-purE | 0.43              | -6.75     | 2.63                    | -8.27            | -9.81            |
| ECK3763-ilvD | -1.41             | -11.72    | 0.34                    | -0.69            | 0.32             |
| ECK3766-ilvC | -0.58             | -8.53     | -0.81                   | 0.06             | 0.51             |
| ECK3762-ilvE | -0.01             | -11.93    | 0.02                    | -0.68            | -0.46            |
| ECK3764-ilvA | -0.15             | -11.55    | -0.84                   | -0.04            | -0.19            |
```

**Discretize the input data**

Quantitative data can be transformed into categorical data using either the BinaryConvert() or TernaryConvert() function, which produce output in a binary (0,1) or ternary (-1,0,1) form, respectively. The threshold argument is used to specify the phenotypic score cutoff used to select strains with a phenotype that is significantly different from that of the designated control, which is usually the phenotype of the wildtype.
For example:

```r
> ter_phenotype <- ternary_convert(matrix=phenotype_data, threshold =0.5)
> head(ter_phenotype)
```

| STREPTOMYCIN.0.05 | SUCINATE | SULFAMONOMETHOXINE.100 | TAUROCHOLATE.0.1 | TAUROCHOLATE.5.5 |
|-------------------|----------|-------------------------|------------------|-----------------|
| ECK3997-purD      | -1       | 1                       | -1               | -1              |
| ECK0516-purE      | 0        | -1                      | 1                | -1              |
| ECK3763-llvD      | 1        | -1                      | 0                | -1              |
| ECK3766-llvC      | -1       | 1                       | -1               | 0               |
| ECK3762-llvE      | 0        | -1                      | 0                | -1              |
| ECK3764-llvA      | 0        | -1                      | -1               | 0               |

```r
> binary_phenotype <- binary_convert (matrix=phenotype_data, threshold =0.5)
> head(binary_phenotype)
```

| STREPTOMYCIN.0.05 | SUCINATE | SULFAMONOMETHOXINE.100 | TAUROCHOLATE.0.1 | TAUROCHOLATE.5.5 |
|-------------------|----------|-------------------------|------------------|-----------------|
| ECK3997-purD      | 1        | 1                       | 1                | 1               |
| ECK0516-purE      | 0        | 1                       | 1                | 1               |
| ECK3763-llvD      | 1        | 0                       | 0                | 1               |
| ECK3766-llvC      | 1        | 1                       | 1                | 0               |
| ECK3762-llvE      | 0        | 1                       | 0                | 1               |
| ECK3764-llvA      | 0        | 1                       | 1                | 0               |

**Calculate similarities/distances between phenotypic profiles**

The pairwise similarity/distance between phenotypic profiles can be calculated by a variety of functions, such as Hamming distance, Pearson Correlation Coefficient, Spearman Correlation Coefficient, Mutual Information. The method selected will depend on the type of assay used.

The function implemented in this package is hamming distance:

```r
> hamming_dist<-hamming_distance(head(ter_phenotype))
> hamming_dist
```

| ECK3997-purD | 0 | 1 | 2 | 3 | 3 | 4 |
|-------------|---|---|---|---|---|---|
| ECK0516-purE| 1 | 0 | 3 | 4 | 2 | 3 |
| ECK3763-llvD| 2 | 3 | 0 | 3 | 1 | 3 |
| ECK3766-llvC| 3 | 4 | 3 | 0 | 4 | 2 |
| ECK3762-llvE| 3 | 2 | 1 | 4 | 0 | 2 |
| ECK3764-llvA| 4 | 3 | 3 | 2 | 2 | 0 |

Example for a self-defined function not included in this package. Pearson Correlation Coefficient was chosen because it's widely used in high-throughput microbial phenotype data:

```r
> pearson_dist=1-corr(t(phenotype_data),method="pearson")
> pearson_dist
```

| ECK3997-purD | 0.00000000  | 0.01130833 | 0.67443135 | 0.792573199 | 0.614310165 | 0.709983271 |
|-------------|-------------|------------|------------|-------------|-------------|-------------|
| ECK0516-purE| 0.01130833  | 0.00000000 | 0.78293109 | 0.891990335 | 0.706927257 | 0.800223580 |
| ECK3763-llvD| 0.78293109  | 0.792573199| 0.00000000 | 0.15107990  | 0.01252945  | 0.017520039 |
| ECK3766-llvC| 0.78293109  | 0.792573199| 0.15107990 | 0.00000000  | 0.017882605 | 0.006075217 |
| ECK3762-llvE| 0.61431017  | 0.709983271| 0.01252945 | 0.017882605 | 0.00000000  | 0.005903022 |
| ECK3764-llvA| 0.709983271 | 0.800223580| 0.017520039| 0.006075217 | 0.005903022 | 0.000000000 |
Parse gene annotations

In order to link the functional annotations associated with a gene to the phenotypic profile of a strain where that gene is mutated, three functions, attr_list(), one_attr() and generate_pairs_similarity_coannotation(), are provided where:

attr_list() takes in a table that has mapping from mutant ids to their annotations and form a list from such mapping, for example, if starting from inputting the strain names and corresponding annotations:

attr_list() creates a list takes where each strain is associated with all the functional annotations and creates a list. To illustrate this, we will create a table with the name “name_attribute”:

```r
> name_attribute = data.frame(id=c("ECK3762-ilvE", "ECK3762-ilvE", "ECK3762-ilvE", "ECK3762-ilvE", "ECK3762-ilvE", "ECK3762-ilvE", "ECK3762-ilvE", "ECK3764-ilvA", "ECK3764-ilvA", "ECK3764-ilvA", "ECK3997-purD", "ECK3997-purD", "ECK3997-purD", "ECK3997-purD", "ECK0516-purE", "ECK0516-purE", "ECK0516-purE", "ECK3763-ilvD", "ECK3763-ilvD", "ECK3763-ilvD", "ECK3766-ilvC", "ECK3766-ilvC", "ECK3766-ilvC", "ECK3766-ilvC"),
   annot=c("ALANINE-VALINESYN-PWY", "THREOCAT-PWY", "ALL-CHORISMATE-PWY", "PWYO-1061", "PHESYN", "ILEUSYN-PWY", "LEUSYN-PWY", "VALSYN-PWY", "COMPLETE-ARO-PWY", "BRANCHED-CHAIN-AA-SYN-PWY", "THREOCAT-PWY", "ILEUSYN-PWY", "BRANCHED-CHAIN-AA-SYN-PWY", "PRPP-PWY", "DENOVOPURINE2-PWY", "PWY-6121", "PWY-6122", "PWY-6277", "PRPP-PWY", "PWY-6123", "DENOVOPURINE2-PWY", "THREOCAT-PWY", "ILEUSYN-PWY", "VALSYN-PWY", "BRANCHED-CHAIN-AA-SYN-PWY", "THREOCAT-PWY", "PANTO-PWY", "ILEUSYN-PWY", "VALSYN-PWY", "PANTOSYN-PWY", "BRANCHED-CHAIN-AA-SYN-PWY")
```

```r
> name_attribute

| ids         | PWY                        |
|-------------|----------------------------|
| ECK3762-ilvE| ALANINE-VALINESYN-PWY      |
| ECK3762-ilvE| THREOCAT-PWY               |
| ECK3762-ilvE| ALL-CHORISMATE-PWY         |
| ECK3762-ilvE| PWYO-1061                  |
| ECK3762-ilvE| VALSYN-PWY                 |
| ECK3762-ilvE| LEUSYN-PWY                 |
| ECK3762-ilvE| COMPLETE-ARO-PWY          |
| ECK3762-ilvE| BRANCHED-CHAIN-AA-SYN-PWY |
| ECK3764-ilvA| THREOCAT-PWY               |
| ECK3764-ilvA| ILEUSYN-PWY                |
| ECK3764-ilvA| BRANCHED-CHAIN-AA-SYN-PWY |
| ECK3997-purD| PRPP-PWY                   |
| ECK3997-purD| DENOVOPURINE2-PWY          |
| ECK3997-purD| PWY-6121                   |
| ECK3997-purD| PWY-6122                   |
| ECK3997-purD| PWY-6277                   |
| ECK0516-purE| PRPP-PWY                   |
| ECK0516-purE| DENOVOPURINE2-PWY          |
| ECK0516-purE| PWY-6123                   |
| ECK3763-ilvD| THREOCAT-PWY               |
| ECK3763-ilvD| ILEUSYN-PWY                |
| ECK3763-ilvD| VALSYN-PWY                 |
```
> attr_list(name_attribute)

```r
$'ECK3762-ilvE'
[1] "ALANINE-VALINESYN-PWY" "THREOCAT-PWY" "ALL-CHORISMATE-PWY" "PYO-1061" "PHESYN" "ILEUSYN-PWY"  
[7] "ILEUSYN-PWY" "VALSYN-PWY" "COMPLETE-ARO-PWY" "BRANCHED-CHAIN-AA-SYN-PWY"

$'ECK3764-ilvA'
[1] "THREOCAT-PWY" "ILEUSYN-PWY" "BRANCHED-CHAIN-AA-SYN-PWY"

$'ECK3997-purD'
[1] "PRPP-PWY" "DENOVOPURINE2-PWY" "PY-6121" "PY-6122" "PY-6277"

$'ECK0516-purE'
[1] "PRPP-PWY" "PY-6123" "DENOVOPURINE2-PWY"

$'ECK3763-ilvD'
[1] "THREOCAT-PWY" "ILEUSYN-PWY" "VALSYN-PWY" "BRANCHED-CHAIN-AA-SYN-PWY"

$'ECK3766-ilvC'
[1] "THREOCAT-PWY" "PANTO-PWY" "ILEUSYN-PWY" "VALSYN-PWY" "PANTOSYN-PWY" "BRANCHED-CHAIN-AA-SYN-PWY"
```

one_attr() takes the output from attr_list() as the input. It generates a table that contains all possible combination of mutants and whether they share annotations (0 stands for not having any same annotations, 1 for having at least 1 same annotation). For example:

```r
> attribute_list=attr_list(name_attribute)
> one_attr(attribute_list)

| mutant1 | mutant2 | sameORnot |
|---------|---------|-----------|
| 1 ECK3762-ilvE ECK3764-ilvA | 1 |
| 2 ECK3762-ilvE ECK3997-purD | 0 |
| 3 ECK3762-ilvE ECK0516-purE | 0 |
| 4 ECK3762-ilvE ECK3763-ilvD | 1 |
| 5 ECK3762-ilvE ECK3764-ilvC | 1 |
| 6 ECK3764-ilvA ECK3997-purD | 0 |
| 7 ECK3764-ilvA ECK0516-purE | 0 |
| 8 ECK3764-ilvA ECK3763-ilvD | 1 |
| 9 ECK3764-ilvA ECK3764-ilvC | 1 |
| 10 ECK3997-purD ECK0516-purE | 1 |
| 11 ECK3997-purD ECK3763-ilvD | 0 |
| 12 ECK3997-purD ECK3764-ilvC | 0 |
| 13 ECK0516-purE ECK3763-ilvD | 0 |
| 14 ECK0516-purE ECK3764-ilvC | 0 |
| 15 ECK3763-ilvD ECK3766-ilvC | 1 |
```

generate_pairs_similarity_coannotation() generates the pairwise similarity table that contains the strain pairs, similarity/distance value and a Boolean column of whether the corresponding strain pairs share the same annotation(s). It takes phenotype data, the result from attr_list() and a function as an argument to specify the similarity metric. For example:

```r
> attribute_list<-one_attr(name_attribute)
> names(attribute_list)<-rownames(phenotype_data) # (Must do) Synchronize the strain names
> generate_pairs_similarity_coannotation(data= phenotype_data,attribute_list=attribute_list, +  dist_metric=pcc_dist)
```
### Get metrics derived of the confusion matrix

After the similarity and co-annotation columns are computed as above, `get_confusionMatrix()` can be used to get the similarity-based confusion matrices and the derived metrics: sensitivity, specificity, precision, and accuracy. `get_confusionMatrix()` also permute the co-annotation column and calculate the confusion matrices and other derived metrics as negative controls. For example:

```r
> new <- generate_pairs_similarity_coannotation(data=phenotype_data, attribute_list=attribute_list, dist_metric=pcc_dist)
```

```r
> get_confusionMatrix_and_metrics(df=new, annot="same_annot", similarity="similarity")
```
Plot the results
graph_corr_annot () takes the output from get_confusionMatrix() to plot the final result, where precisions were plotted against the ranked pairs of mutants. Enrichment for sensitivity, specificity, precision, and accuracy could be compared with the dotted line, which is the negative control. For example:

```r
> confusionMatrix_obj <- get_confusionMatrix_and_metrics (new,"same_annot","similarity",seed=103)
> metric="precision"; similarity="pcc"; subset=dim(confusionMatrix_obj)[1]; cols="blue";
> ylim=c(0,1); lwd=1 # set graphing parameters
> graph_corr_annot(confusionMatrix_obj, metric, similarity, subset, cols, x_lab="", ylim=c(0,1.05), lwd)
```

Enrichment for co-annotations

> metric="sensitivity" # use another metric for the y axis
> graph_corr_annot(confusionMatrix_obj, metric, similarity, subset, cols, x_lab="", ylim=c(0,1.05), lwd)
**Helper functions**

The following 10 functions are intended to speed up the phenotype data curation process:

any_incomplete() checks if the input matrix, dataframe or datatable contains any NA, NAN, NULL or "" (empty string). For example:

```
> incomplete_phenotype_data <- phenotype_data
> incomplete_phenotype_data[1:2, 1:2] <- NA  # introduce some NA
> any_incomplete(incomplete_phenotype_data)
```

```

|                  | Dimension: 6 rows * 5 columns |
|------------------|-------------------------------|
| any_incomplete() |                               |
| na               |                                |
| null             | named integer(0)              |
| nan              | named integer(0)              |
```

---

![Enrichment for co-annotations](image-url)
$empty$

named integer(0)

$completeness$

[1] "4 of NA, NAN, NULL, or empty character is found from 30 data points. They constitute 13.3333333333333%"

filter_table() filters the input matrix, dataframe or datatable so that all rows and columns with NA/NAN/NULL/"" are removed. For example:

```r
> filter_table(incomplete_phenotype_data)
```

|                  | SULFAMONOMETHOXINE.100 | TAUROCHOLATE.0.1 | TAUROCHOLATE.0.5 |
|------------------|-------------------------|------------------|------------------|
| ECK3997-purD     | 3.33                    | -13.46           | -13.79           |
| ECK3763-ilvD     | 0.34                    | -0.69            | 0.32             |
| ECK3764-ilvC     | -0.81                   | 0.06             | 0.51             |
| ECK3762-ilvE     | 0.02                    | -0.68            | -0.46            |
| ECK3766-ilvA     | -0.84                   | -0.04            | -0.19            |

graph_table() takes a matrix, dataframe or a table as the input and represents it using a heatmap. It also deals with continuous/categorical/mixed variables. For example:

```r
> graph_table(phenotype_data)
```

![Heatmap of phenotype_data](image1)

```r
> graph_table(ter_phe_notype)
```

![Heatmap of ter_phe_notype](image2)
checkDuplicates_vect() checks if an input vector has duplicates. If so, it will return the frequency table. Otherwise, the text “Everything in this vector is unique” will be returned. For example:

```r
> my_vector <- c(1,1,2,2,3)
> checkDuplicates_vect(my_vector)
[1] "Some duplicates are found:"
  vect
  1 2 3
  2 2 1

> my_vector <- c(1,2,3)
> checkDuplicates_vect(my_vector)
[1] "Everything in this vector is unique"
```

change_names() changes row names or column names of a matrix, dataframe or datatable based on another matrix/dataframe/datatable. For example:

```r
> new_col_names <- c("0.05 µg/ml streptomycin",
                    "0.3% succinate",
                    "100 µg/ml sulfamonomethoxine","0.1% taurocholate","0.5% taurocholate ")

> change_names(rowOrCol="col", phenotype_data, matrix(c(colnames(phenotype_data), new_col_names), ncol=2, byrow=FALSE))
```

melt_similarity() takes a similarity matrix or a distance object as the input and converts it to a long form dataframe, with an option to sort the molten dataframe by the 3rd column (the numeric column). For example:

```r
> dist(phenotype_data) # the distance object of interest

> melt_similarity(dist(phenotype_data))
```
When compared with reshape2::melt(), the differences are:

1. **melt_similarity()**
   - Can take a distance object as the main input
2. When a matrix is used as the main input, it has to be a similarity matrix
3. **melt_similarity()** remove the diagonal elements and the duplicated pairs that share the same similarity.

**convert_table()** takes a matrix, dataframe or a datatable as the input and converts the types of elements to a designated type. For example:

```R
> str(phenotype_data)
'data.frame': 6 obs. of 5 variables:
$ STREPTOMYCIN.0.05  : num 1.48 0.43 -1.41 -0.58 -0.01 -0.15 ...
$ SUCCINATE            : num 12.3 -6.75 -11.72 -8.53 -11.93 ...
$ SULFAMONOMETHOXINE.100: num 3.33 2.63 0.34 -0.81 0.02 -0.84 ...
$ TAUCROCHOLATE.0.1   : num 13.46 -8.27 -0.69 0.06 -0.68 ...
$ TAUCROCHOLATE.0.5   : num 13.79 -9.81 0.32 0.51 -0.46 ...

> phenotype_data_chr <- convert_table(phenotype_data, as.character)
> str(phenotype_data_chr)
'data.frame': 6 obs. of 5 variables:
$ STREPTOMYCIN.0.05  : chr "1.48" "0.43" "-1.41" "-0.58" ...
$ SUCCINATE            : chr "12.3" "-6.75" "-11.72" "-8.53" ...
$ SULFAMONOMETHOXINE.100: chr "3.33" "2.63" "0.34" "-0.81" ...
$ TAUCROCHOLATE.0.1   : chr "13.46" "-8.27" "-0.69" "0.06" ...
$ TAUCROCHOLATE.0.5   : chr "13.79" "-9.81" "0.32" "0.51" ...
```

**remove_NA()** removes NA from an R vector object. For example:

```R
> remove_NA(c(1,2,3,4,NA,NA))
[1] 1 2 3 4
```

**Summary**

The microbialPhenotypes package has implemented a pipeline to systematically parse and analyze high-throughput phenotype screens, with many functions as well as an example using a published *E. coli* dataset (Nichols et al., 2011). Although the motivation for this software is to process microbial phenotype data, we expect its usability to be easily extended to multivariate dataset with distinct annotation sets.

**Acknowledgements**

This work was supported by a grants from the National Science Foundation Division of Biological Infrastructure [1458400] and the National Institutes of Health [R01GM089636, U41HG008735]. PW wrote the package and drafted the manuscript. DS, JH supervised the project as well as testing the usability of the code.
Bibliography

Alonezi, S., Tusiimire, J., Wallace, J., Dufton, M. J., Parkinson, J. A., Young, L. C., . . . Watson, D. G. (2016). Metabolomic Profiling of the Effects of Melittin on Cisplatin Resistant and Cisplatin Sensitive Ovarian Cancer Cells Using Mass Spectrometry and Biolog Microarray Technology. *Metabolites, 6*(4). doi:10.3390/metabo6040035

Alonezi, S., Tusiimire, J., Wallace, J., Dufton, M. J., Parkinson, J. A., Young, L. C., . . . Watson, D. G. (2017). Metabolomic Profiling of the Synergistic Effects of Melittin in Combination with Cisplatin on Ovarian Cancer Cells. *Metabolites, 7*(2). doi:10.3390/metabo7020014

Collins, S. R., Schuldiner, M., Krogan, N. J., & Weissman, J. S. (2006). A strategy for extracting and analyzing large-scale quantitative epistatic interaction data. *Genome Biol, 7*(7), R63. doi:10.1186/gb-2006-7-7-r63

Deng, Y., Gao, L., Wang, B., & Guo, X. (2015). HPOSim: an R package for phenotypic similarity measure and enrichment analysis based on the human phenotype ontology. *PLoS One, 10*(2), e0115692. doi:10.1371/journal.pone.0115692

Kritikos, G., Banzhaf, M., Herrera-Dominguez, L., Koumouts, A., Wartel, M., Zietek, M., & Typas, A. (2017). A tool named Iris for versatile high-throughput phenotyping in microorganisms. *Nat Microbiol, 2*, 17014. doi:10.1038/nmicrobiol.2017.14

Nichols, R. J., Sen, S., Choo, Y. J., Beltrao, P., Zietek, M., Chaba, R., . . . Gross, C. A. (2011). Phenotypic landscape of a bacterial cell. *Cell, 144*(1), 143-156. doi:10.1016/j.cell.2010.11.052

Price, M. N., Wetmore, K. M., Waters, R. J., Callaghan, M., Ray, J., Liu, H., . . . Deutschbauer, A. M. (2018). Mutant phenotypes for thousands of bacterial genes of unknown function. *Nature, 557*(7706), 503-509. doi:10.1038/s41586-018-0124-0

Rishi, H. S., Toro, E., Liu, H., Wang, X., Qi, L. S., & Arkin, A. P. (2020). Systematic genome-wide querying of coding and non-coding functional elements in E. coli using CRISPRi. doi:10.1101/2020.03.04.975888

Tohsato, Y., & Mori, H. (2008). Phenotype profiling of single gene deletion mutants of E. coli using Biolog technology. *Genome Inform, 21*, 42-52. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/19425146

Vaas, L. A., Sikorski, J., Hofner, B., Fiebig, A., Buddruhs, N., Klenk, H. P., & Goker, M. (2013). opm: an R package for analysing OmniLog(R) phenotype microarray data. *Bioinformatics, 29*(14), 1823-1824. doi:10.1093/bioinformatics/btt291

Vehkala, M., Shubin, M., Connor, T. R., Thomson, N. R., & Corander, J. (2015). Novel R pipeline for analyzing Biolog Phenotypic MicroArray data. *PLoS One, 10*(3), e0118392. doi:10.1371/journal.pone.0118392

Vuckovic, D., Gasparini, P., Soranzo, N., & lotchkova, V. (2015). MultiMeta: an R package for meta-analyzing multi-phenotype genome-wide association studies. *Bioinformatics, 31*(16), 2754-2756. doi:10.1093/bioinformatics/btv222

Wetmore, K. M., Price, M. N., Waters, R. J., Lamson, J. S., He, J., Hoover, C. A., . . . Deutschbauer, A. (2015). Rapid quantification of mutant fitness in diverse bacteria by sequencing randomly bar-coded transposons. *MBio, 6*(3), e00306-00315. doi:10.1128/mBio.00306-15
Peter I-Fan Wu  
Department of Biochemistry & Biophysics, Texas A&M University  
College Station, Texas  
US  
peterwu19881230@tamu.edu

James C. Hu  
Department of Biochemistry & Biophysics, Texas A&M University  
College Station, Texas  
US  
deceased

Deborah A. Siegele  
Department of Biology, Texas A&M University  
College Station, Texas  
US  
siegele@bio.tamu.edu