User Manual for AutoMATA

1. Introduction

AutoMATA is a user-friendly analysis platform designed for the processing and analysis of gene, mRNA, and protein expression data. Tailored for multi-omics studies in *Homo sapiens*, *Mus musculus*, *Bos taurus*, and *Drosophila melanogaster*, AutoMATA streamlines the analysis pipeline from raw data to biological insights. The platform integrates twelve deep learning models alongside sixteen analytical modules, providing a flexible framework for model training, interpretation, and downstream biological analysis. AutoMATA is organised into three major components: (A) the Data Processing Module, (B) the Deep Learning Module, and (C) the Data Analysis Module. Together, these three modules form a cohesive workflow that bridges multi-omics data processing, predictive modelling, and bioinformatics analysis. AutoMATA is freely available at http://automata.biotools.bio/, and its source code is provided on GitHub at https://github.com/ABILiLab/AutoMATA.

2. Environment

To simplify the installation process and reduce the complexity in environment configuration, AutoMATA provides pre-packaged environment files in .yaml format. Users can set up the required environments by executing a single command in the terminal. Additionally, due to hardware limitations, AutoMATA recommends users upload files smaller than 200MB and ideally expect processing times not exceeding 2 hours to maintain AutoMATA's normal operation.

AutoMATA’s runtime environment is divided into two parts based on its functional modules:

- Model Training and Application Environment

  This environment is developed based on Python and includes essential packages such as PyTorch (1.10.1), scikit-learn (0.24.2), pandas (1.1.5), and numpy (1.19.2).

  To install this environment, use the following command:

```
conda env create -f environment.yaml
```

- Data Analysis Environment

  This environment is based on the R language. The environment configuration file is "environment_R.yaml", and the installation command is:

```
conda env create -f environment_R.yaml
```

  Then, there are also some R packages that need to be installed by running R scripts. First, activate your configured conda environment:

```
conda activate env_name
```

  (Replace “env_name” with the name of your environment.)

  Run the script ‘install_packages.R’:

```
Rscript install_packages.R
```

3. The workflow of AutoMATA

This section provides a step-by-step guide to the overall workflow of AutoMATA, which is organised into three models: (A) the Data Processing Module, (B) the Deep Learning Module, and (C) the Data Analysis Module.

The Deep Learning Module incorporates 12 deep learning (DL) models for both model training and application. These models include:

- Convolutional Neural Network (CNN)
- Autoencoder
- Long Short-Term Memory (LSTM)
- Multilayer Perception (MLP)
- Recurrent Neural Network (RNN)
- Radial Basis Function Neural Network (RBFNN)
- Self-Organising Map (SOM)
- Transformer
- Ladder Network
- Pseudo Labelling

- Variational Autoencoder (VAE)
- DeepCluster

The Data Analysis Module provides a range of bioinformatics analysis functions in omics data implemented in R. This module encompasses a variety of functions for differential expression analysis, data visualisation, and functional enrichment. Key features include:

- Correlation heatmap
- Differential gene cluster heatmap
- Principal component analysis (PCA)
- Venn diagrams (VENN)
- Volcano plots with Gene Set Enrichment Analysis (GSEA)
- Gene Ontology (GO) enrichment analysis
- Kyoto Encyclopedia of Genes and Genomes (KEGG) pathway enrichment
- Dumbbell and bar plots (Dumbbell_Bar)
- DESeq2 for read count-based expression data
- limma for FPKM-based expression analysis
- Protein-Protein Interaction Network analysis

3.1 Model training

For the Deep Learning Module in AutoMATA, we provide two flexible training strategies to accommodate different research needs and ensure robust model performance: the traditional train-validation-test split and stratified k-fold cross-validation. Users can select their preferred approach, upload the corresponding data, and customise the training process to suit their specific analytical goals. AutoMATA allows fine-tuning several key hyperparameters, including the number of epochs, learning rate, early stopping patience, and batch size. Additionally, users can specify the loss function and optimiser based on the distribution and nature of their dataset. The number of output classes can be set between 2 and 7 to support binary and multi-class classification tasks.

Environment Setup

To begin model training, users must first configure the Python-based conda environment as described in the “Environment” section. After activating the environment, navigate to the appropriate directory and execute the training script via the command line using the desired parameters.

Script Parameters

- --kfold: Integer; set to 0 to disable k-fold, or to a number such as 3 to enable stratified k-fold cross-validation with three folds.
- --ratio: String; defines dataset split (e.g., "8:1:1" for train/validation/test). Set to 0 to disable.
- --epochs: Number of training epochs.
- --es: Early stopping patience. Set equal to epochs to disable early stopping.
- --lr: Learning rate.
- --bs: Batch size.
- --loss_function: Choose from crossentropy, nllloss, or focalloss.
- --optimizer_function: Choose from adam, rmsprop, or sgd.
- --output_size: Number of output classes, the range is [2, 7].
- --random_seed: Keep the same random seed to ensure reproducibility.

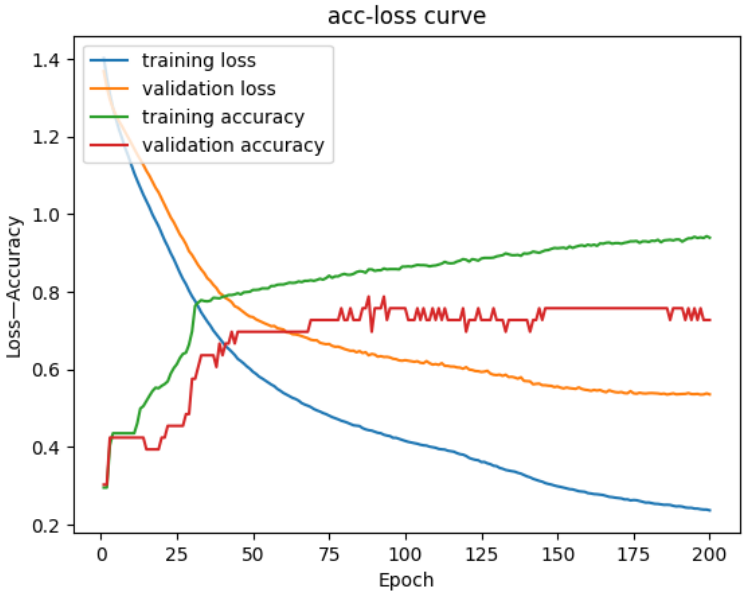


Figure 1. Example of accuracy-loss curves of the RBFNN model after hyperparameter tuning.

Data Format

The training data should be in tab-separated (.txt) format. The first column must contain sample names, and the last column should be labelled "Label" and contain integer class labels ranging from 0 to 5. The first row must contain column headers.

Example Commands

Activate your configured conda environment:

```
conda activate env_name
```

(Replace “env_name” with the name of your environment.)

Navigate to the training directory:

```
cd code/train
```

Run model training using the following commands:

- Autoencoder

```
python autoencoder.py --kfold 0 --ratio 0 --epochs 50 --es 10 --lr 0.01 --bs 32 --loss_function crossentropy --optimizer_function adam --output_size 2
```

- CNN

```
python cnn.py --kfold 0 --ratio 8:1:1 --epochs 50 --es 10 --lr 0.01 --bs 32 --loss_function crossentropy --optimizer_function adam --output_size 4
```

- LSTM

```
python lstm.py --kfold 4 --ratio 0 --epochs 50 --es 10 --lr 0.005 --bs 32 --loss_function nllloss --optimizer_function rmsprop --output_size 2 –random_seed 42
```

- MLP

```
python mlp.py
```

You may also adjust internal model parameters such as “hidden_size_1” to match the distribution of your dataset, or “dropout_rate” to prevent overfitting.

3.2 Model application

The Model Application functionality within the Deep Learning Module enables users to apply trained models to predict unknown samples – an essential step for downstream biological discovery. Building upon the previously trained models, users can select a model type, upload a trained model, and provide a testing dataset. AutoMATA will then generate two output files: one reporting the model's testing performance and another containing predicted labels for unknown samples, thereby addressing predictive needs in multi-omics research.

Usage Instructions

After completing model training and setting up the appropriate conda environment (as described in the "Model training" section), users can apply their trained models using the script commands outlined below.

Script Parameters

- --bs: Batch size.
- --model_type: Specify one of the supported model types: CNN, AutoEncoder, LSTM, MLP, RBFN, RNN, or Transformer.
- --model_path: Path to the trained model file (e.g., model.pth).
- --model_autoencoder_path: Required only when the 'model_type' is 'AutoEncoder'; specify the path to the saved 'model_autoencoder.pth' file.

Input Data Format

The input test file should be tab-delimited (.txt) with:

- The first column: Sample names
- The last column: Ground truth labels (integer values ranging from 0 to 5)
- The first row: Column headers; the last column must be named "Label"

Example Commands

1. Activate your conda environment:

```
conda activate env_name
```

(Replace 'env_name' with your environment name.)

2. Navigate to the application directory:

```
cd code/application
```

3. Run the model application:

To apply CNN (default parameters):

```
python general.py
```

To apply the Transformer:

```
python general.py --model_type Transformer --model_path ./model/model.pth --bs 16
```

To apply the AutoEncoder:

```
python general.py --model_type AutoEncoder
--model_path ./model/model_cls.pth
--model_autoencoder_path ./model/model_autoencoder.pth --bs 8
```

To apply SOM ():

```
python som.py
```

3.3 Data analysis

The Data Analysis Module provides a robust suite of tools for differential expression analysis and various downstream bioinformatics analyses. This module includes commonly used and advanced visualisation techniques such as:

- Correlation heatmaps
- Principal Component Analysis (PCA)
- Volcano plots integrated with Gene Set Enrichment Analysis (GSEA)
- Differential gene cluster heatmaps
- Venn diagrams in three styles: classic Venn, VennPie, and barplot

- Functional enrichment analysis, including:
    - Gene Ontology (GO) enrichment visualised in five styles: bubble, barplot, chord, cluster, and circle
    - KEGG pathway enrichment visualised in three formats: bubble, chord, and cluster
- Protein-Protein Interaction Network plots in three styles: linear, kk, and stress.
- Dumbbell-bar hybrid plots for comparative expression analysis

Additionally, this module provides flexible image export options suitable for publication and presentation purposes. Moreover, we recommend that users employ reasonable and biologically relevant statistical thresholds and cross-reference results with established biological knowledge before drawing conclusions. We suggest that users visually inspect their data using PCA plots or other methods available in AutoMATA's visualization module to identify any obvious batch-related clustering before proceeding with integrated analysis.

Preparatory Work

Before running any analysis:

1. Set up the R environment as described in the "Environment" section.
2. Activate the R conda environment with:

```
conda activate r_env_name
```

(Replace ‘r_env_name’ with your environment name.)

3. Navigate to the analysis directory:

```
cd code/analysis
```

1: Correlation Heatmap

Script: ‘cor_heatmap.r’

This script generates a correlation heatmap from the input expression matrix

Required Parameters:

- -i (input): Path to the input data file (tab-separated .txt file). The first row should contain column names.

Example Commands:

- Run with default settings:

```
Rscript cor_heatmap.r
```

- Run with a specified input file:

```
Rscript cor_heatmap.r -i cor_heatmap_test.txt
```

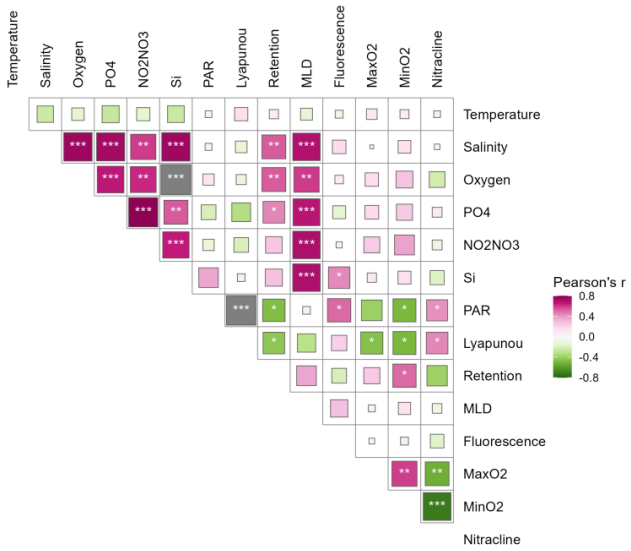


Figure 2. An example of a correlation heatmap.

2: Differential Gene Cluster Heatmap

Script: "df_cluster_heatmap.R"

This script generates a clustered heatmap of differentially expressed genes with optional row and column annotations.

Script Parameters:

- -i (input): Path to the input data file. The first row should contain column names, and the first column should contain row names. The file should be tab-delimited (\t).
- -a (type): Specifies the type of figure to generate, including row and/or column annotations. Choose one of the following:

⇒ data_with_row_col: Includes both row and column annotations.

⇒ data_with_col_annotation: Includes only column annotations (e.g., Group, Age, Grade, Stage, Sex).

⇒ data_with_row_annotation: Includes only row annotations (e.g., Path, Celltype).

⇒ 'only_data': indicates only output heatmap with no annotations.

- -b (show_row_name): Whether to display row names. Input ‘TRUE’ or ‘FALSE’.
- -c (show_col_name): Whether to display column names. Input ‘TRUE’ or ‘FALSE’.
- -d (clustering_dis_row): Method used for calculating row clustering distance. Choose from: ‘correlation’, ‘euclidean’, ‘maximum’, ‘manhattan’, ‘canberra’, ‘binary’, ‘minkowski’.
- -e (clustering_dis_col): Method used for calculating column clustering distance. Same options as above.
- -g (annotation_col_file): Path to the column annotation file. Required if ‘-a’ is set to ‘data_with_row_col’ or ‘data_with_col_annotation’.
- -f (annotation_row_file): Path to the row annotation file. Required if ‘-a’ is set to ‘data_with_row_col’ or ‘data_with_row_annotation’.
- -h (scal): Whether to scale the data. Choose from:

  ⇒ row: Scale by row

  ⇒ column: Scale by column

  ⇒ none: No scaling
- -k (group): Whether to display grouping in the heatmap. Requires the column annotation file to include a ‘group’ column. Input ‘TRUE’ or ‘FALSE’.

Example Commands

Run the script with default parameters:

```
Rscript df_cluster_heatmap.R
```

Run the script with specified parameters:

```
Rscript df_cluster_heatmap.R -i df_gene_cluster_test.txt -a
data_with_col_annotation -g df_gene_cluster_annotation_col.txt -c FALSE -b
TRUE -d correlation -e correlation -h row -k TRUE
```

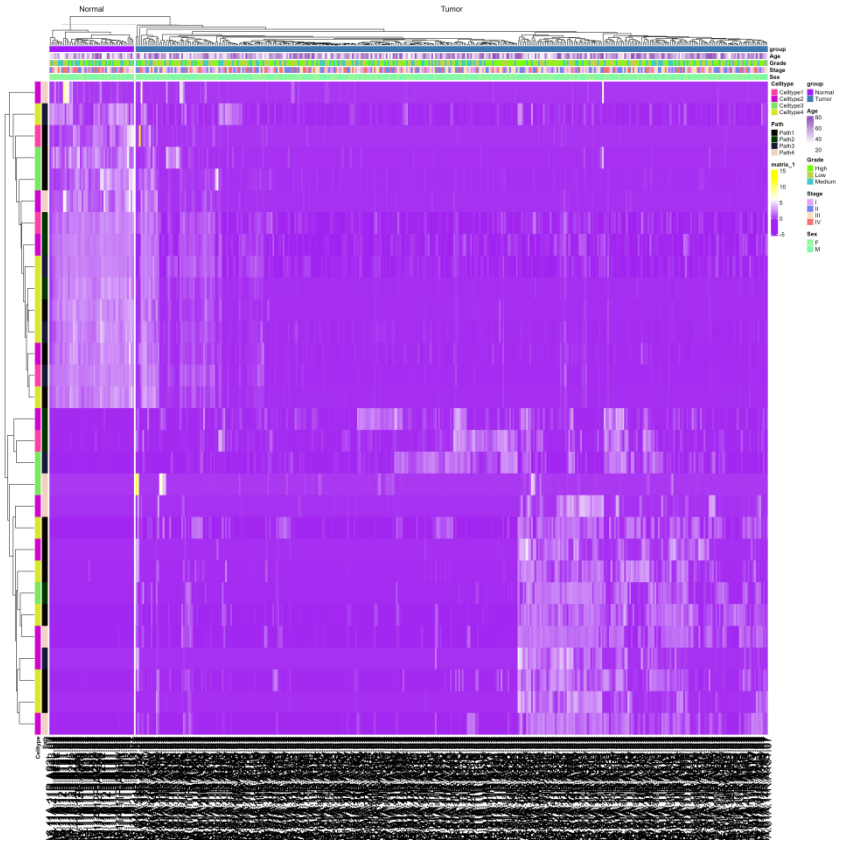


Figure 3. Example output of a differential gene cluster heatmap.

3: PCA

Script Parameters

- -i (input): the path of input data file. The first column of the data is the group information. The seperater is '\t'.
- -c (confidence_level): confidence level.
- -b (boundary): decides whether to add boundary plots, which are the top and right sub-plots of the figure. Please input TRUE or FALSE.

- -p (permanova): decides whether to add PERMANOVA analysis. Please input TRUE or FALSE.
- -m (method): the method for permanova analysis. Please input one of 'manhattan', 'euclidean', 'canberra', 'clark', 'bray', 'kulczynski', 'jaccard', 'gower', 'altGower', 'morisita', 'horn', 'mountford', 'raup', 'binomial', 'chao', 'cao', 'mahalanobis', 'chisq', 'chord', 'hellinger', 'aitchison', and 'robust.aitchison'.

Example Commands

Run the script with default parameters:

```
Rscript pca.R
```

Run the script with specified parameters:

```
Rscript pca.R -i pca_test.txt -c 0.92 -b FALSE -p FALSE
```

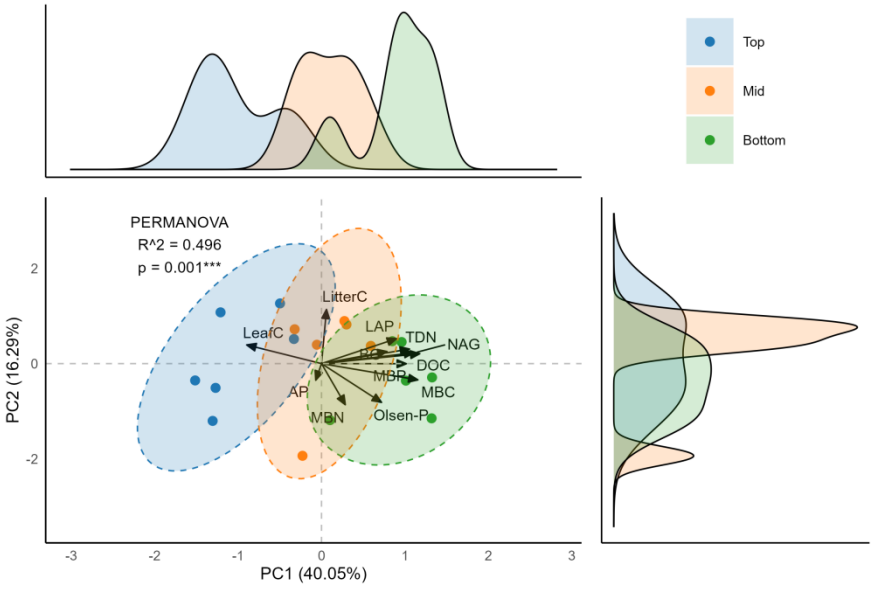


Figure 4. An example of PCA.

4: VENN

Script Parameters

- -i (input): the path of input data file. The first row is the column names, and each column is the information for each group. The seperater is '\t'.

- -t (type): the type of plot. Please input one of Venn, Vennpie and Barplot.

Example Command

Run the script with default parameters:

```
Rscript venn.R
```

Run the script with specified parameters:

```
Rscript venn.R -i venn_test.txt -t Vennpie
```

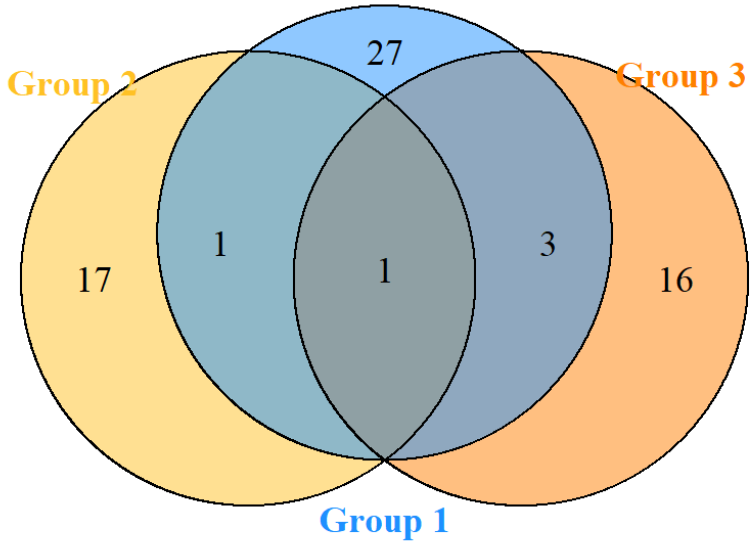


Figure 5. An example of VENN.

5: Volcano with GSEA

Script Parameters

- -i (input): the path of input data file. Keep logFC, padj, gene column names of dataset consistent. The seperater is '\t'.
- -g (gmt): the path of gmt data file. If you want to conduct GSEA analysis, then specialize it to the path gmt data file, otherwise set it to 'none'.
- -a (fc_thr): the threshold of $\log_2$FC. If you wish to obtain stringent results,

use 1.5, 2, or other strict thresholds (|log_2FC| ⩾ 1.5 or ⩾ 2); conversely, use 0.58 or other lenient thresholds (|log_2FC| ⩾ 0.58).

- -b (padj_thr): the threshold of padj (adjusted p-value). If you wish to obtain stringent results, use 0.01 or other strict thresholds (padj < 0.01); conversely, use 0.05 or other lenient thresholds (padj < 0.05).
- -c (top): the number of higher-order genes to be emphasized. The emphasized genes are in the dashed box.
- -d (top_fc_thr): the threshold of log_2FC for the higher-order gene. It should be more strict than 'fc_thr'.
- -e (top_padj_thr): the threshold of padj for the higher-order gene. It should be more strict than 'padj_thr'.
- -f (gene_sig): the list of marked genes, the separator must be comma ','.

Example Commands

Run the script with default parameters:

```
Rscript volcano_gsea_padj.R
```

Run the script with specified parameters:

```
Rscript volcano_gsea_padj.R -i volcano_test.txt -g none -a 1 -b 0.05 -c 30 -d 2 -e 0.01 -f KRAS,FOSL1
```

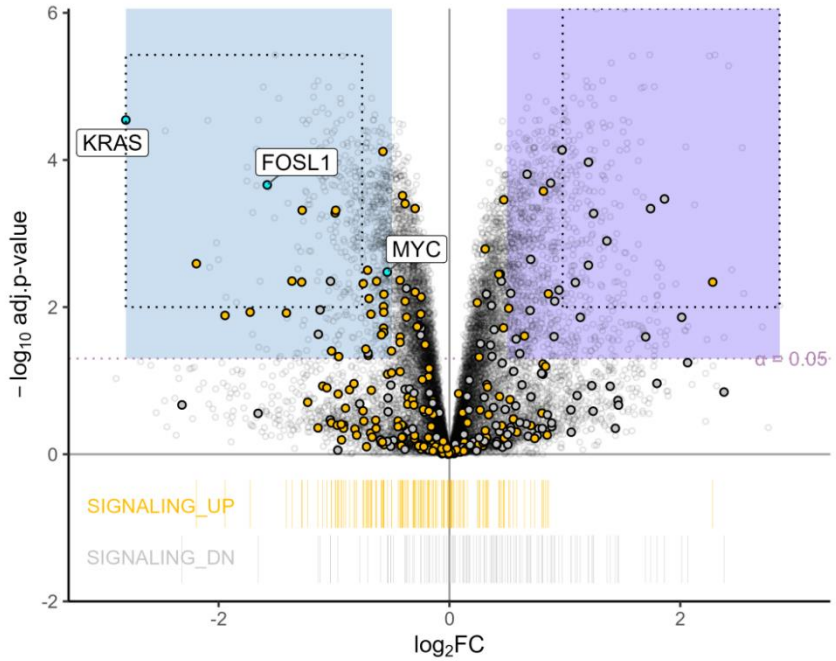


Figure 6. An example of volcano with GSEA.

6: GO Enrichment

Script Parameters

- -i (input): the path of input data file. The first column is gene symbol, and the column name is gene. The data file only needs the gene symbol if 'type' is bubble or bar, otherwise needs to make sure that the data file has numeric data with the column name logFC to sort by the size of the value.
- -a (type): the type of figure. Please input one of bubble, bar, chord, cluster and circle.
- -b (organism): Please input one of Homo_sapiens, Mus_musculus, Bovine, Homo_sapiens and Drosophila_melanogaster.
- -c (pvalue): pvalue threshold for GO enrichment analysis. The strict pvalue threshold is 0.01 (pvalue<0.01), and lenient threshold is 0.05 (pvalue<0.05).
- -d (qvalue): qvalue threshold for GO enrichment analysis. The gold standard is 0.05.
- -e (termNum): the number of terms for each ontology to be displayed.
- -g (adjust): the pvalue adjustment method for GO enrichment analysis. Please input one of holm, hochberg, hommel, bonferroni, BH, BY, fdr, none.

Example Commands

Run the script with default parameters:

```
Rscript go_enrichment.R
```

Run the script with specified parameters:

```
Rscript go_enrichment.R -i go_enrichment_test.txt -a bubble -b Mus_musculus -c 0.01 -e 10 -g BH
```

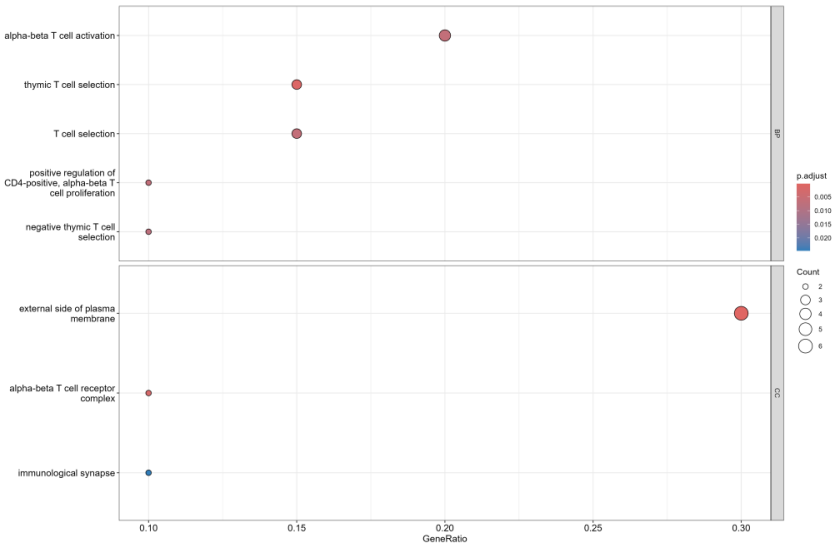


Figure 7. An example of GO Enrichment.

7: KEGG Enrichment

Script Parameters

- -i (input): the path of input data file. The first column is gene symbol, and the column name is gene. The data file only needs the gene symbol if 'type' is bubble, otherwise needs to make sure that the data file has numeric data with the column name logFC to sort by the size of the value.
- -a (type): the type of figure. Please input one of bubble, chord, and cluster.
- -b (organism): Please input one of hsa, mmu, bos, and dme.
- -c (pvalue): pvalue threshold for KEGG enrichment analysis. The strict pvalue threshold is 0.01 (pvalue<0.01), and lenient threshold is 0.05 (pvalue<0.05).
- -d (qvalue): qvalue threshold for KEGG enrichment analysis. The gold standard is 0.05.
- -e (termNum): the number of terms for each ontology to be displayed.
- -g (adjust): the pvalue adjustment method for GO enrichment analysis. Please input one of holm, hochberg, hommel, bonferroni, BH, BY, fdr, none.

Example Commands

Run the script with default parameters:

```
Rscript kegg_enrichment.R
```

Run the script with specified parameters:

```
Rscript kegg_enrichment.R -i kegg_enrichment_test.txt -a bubble -b dme -d 0.01 -e 10 -g BH
```

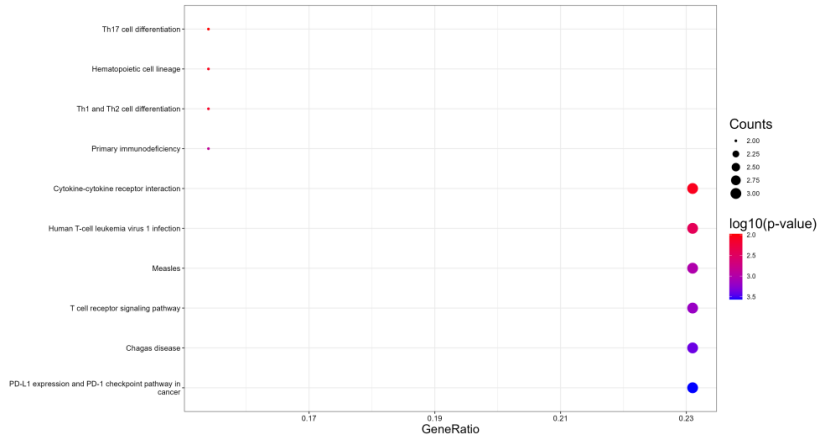


Figure 8. An example of KEGG Enrichment.

8: Dumbbell_Bar

Script Parameters

- -i (input): the path of input data file. This file is used to draw dumbbell diagrams.
- -c (data_bar): the file path of bar plot. This file is used to draw barplot diagrams. Keep the contents and name of the first column in both two files similar and same respectively.
- -a (y_label): the y label of dumbbell_bar plot.
- -b (mark_fams): the terms that will be marked, which are factors in the first column of data files, the separator must be comma ','.

Example Commands

Run the script with default parameters:

```
Rscript dumbbell_bar.R
```

Run the script with specified parameters:

```
Rscript dumbbell_bar.R -i dumbbell_test.txt -c dumbbell_barplot_test.txt -a number -b Notothenioid
```

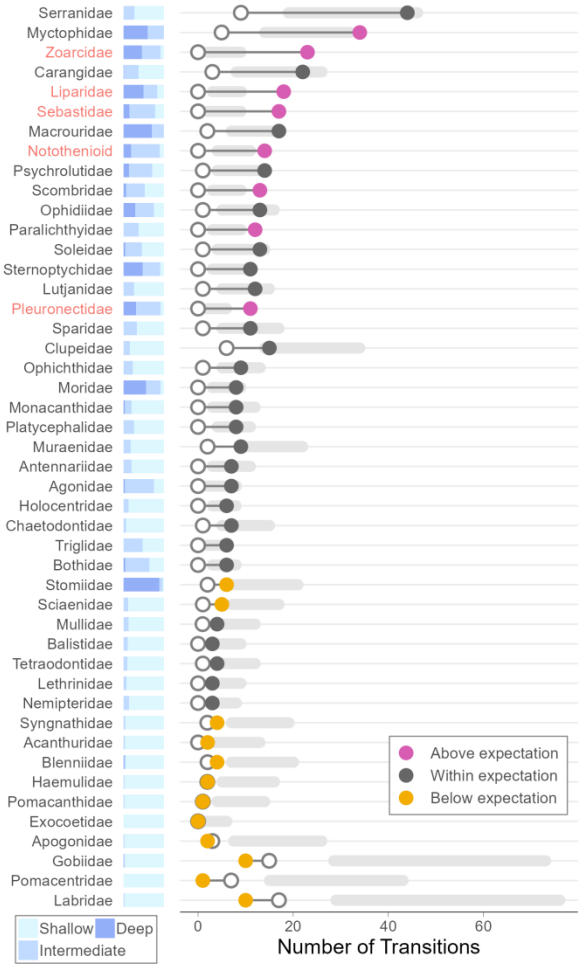


Figure 9. An example of Dumbbell_Bar.

9: DESeq2 for Read Count

Script Parameters

- -i (expression_file): the path of expression file. The first column is row names, the first row is column names.
- -k (info_file): the path of group information file. You need to make sure that the order of the samples in the expression file corresponds to the

group info file order here. Keep the row names in the group file the same as the column names in the expression file: Control_1, Control_2, Treatment_1, Treatment_2. The Group file must contain a Group column, and the value of the group column must be 'Control' or 'Treatment'.

- -c (fc_thr): the log_2FC threshold for differential expression analysis. If you wish to obtain stringent results, use 1.5, 2, or other strict thresholds ($|log_2FC| \geqslant 1.5$ or $\geqslant 2$); conversely, use 0.58 or other lenient thresholds ($|log_2FC| \geqslant 0.58$).
- -d (padj_thr): the padj threshold for differential expression analysis. If you wish to obtain stringent results, use 0.01 or other strict thresholds (padj < 0.01); conversely, use 0.05 or other lenient thresholds (padj < 0.05).
- -e (correction): This argument defines hypothesis correction method. Please input one of none, BH, BY, holm, hochberg, hommel, or bonferroni

Example Commands

Run the script with default parameters:

```
Rscript DESeq2_read_count.R
```

Run the script with specified parameters:

```
Rscript DESeq2_read_count.R -i expression.txt -k info.txt -c 2 -d 0.05 -e BH
```

10: limma for FPKM

Script Parameters

- -i (expression_file): the path of expression file. The first column is row names, the first row is column names.
- -k (info_file): the path of group information file. You need to make sure that the order of the samples in the expression file corresponds to the group info file order here. Keep the row names in the group file the same as the column names in the expression file: Control_1, Control_2, Treatment_1, Treatment_2. The Group file must contain a Group column,

and the value of the group column must be 'Control' or 'Treatment'.

- -c (fc_thr): the log_2FC threshold for differential expression analysis. If you wish to obtain stringent results, use 1.5, 2, or other strict thresholds ($|log_2FC| \geqslant 1.5$ or $\geqslant 2$); conversely, use 0.58 or other lenient thresholds ($|log_2FC| \geqslant 0.58$).
- -d (padj_thr): the padj threshold for differential expression analysis. If you wish to obtain stringent results, use 0.01 or other strict thresholds (padj < 0.01); conversely, use 0.05 or other lenient thresholds (padj < 0.05).
- -e (correction): This argument defines hypothesis correction method. Please input one of none, BH, BY, holm, hochberg, hommel, or bonferroni

Example Commands

Run the script with default parameters:

```
Rscript limma_fpkm_df.R
```

Run the script with specified parameters:

```
Rscript limma_fpkm_df.R -i expression.txt -k info.txt -c 2 -d 0.05 -e BH
```

11: PPI Network

Script Parameters

- -i (input): the path of input data file. The first row is column name, and the first column is gene. The file can be consist of gene symbol, ENTREZID, or ENSEMBL.
- -a (type): this argument is the type of data, please input one of SYMBOL, ENTREZID, and ENSEMBL.
- -b (organism): this argument is the organism, please input one of Mus_musculus, Homo_sapiens, Drosophila_melanogaster, and Bos_taurus.
- -c (score_threshold): the interaction results were filtered according to the protein interaction scores.

- -d (plot_type): this parameter is the type of plot, please input one of linear, kk, and stress.
- -e (show_num): only genes with more than <show_num> nodes are shown in the plot.

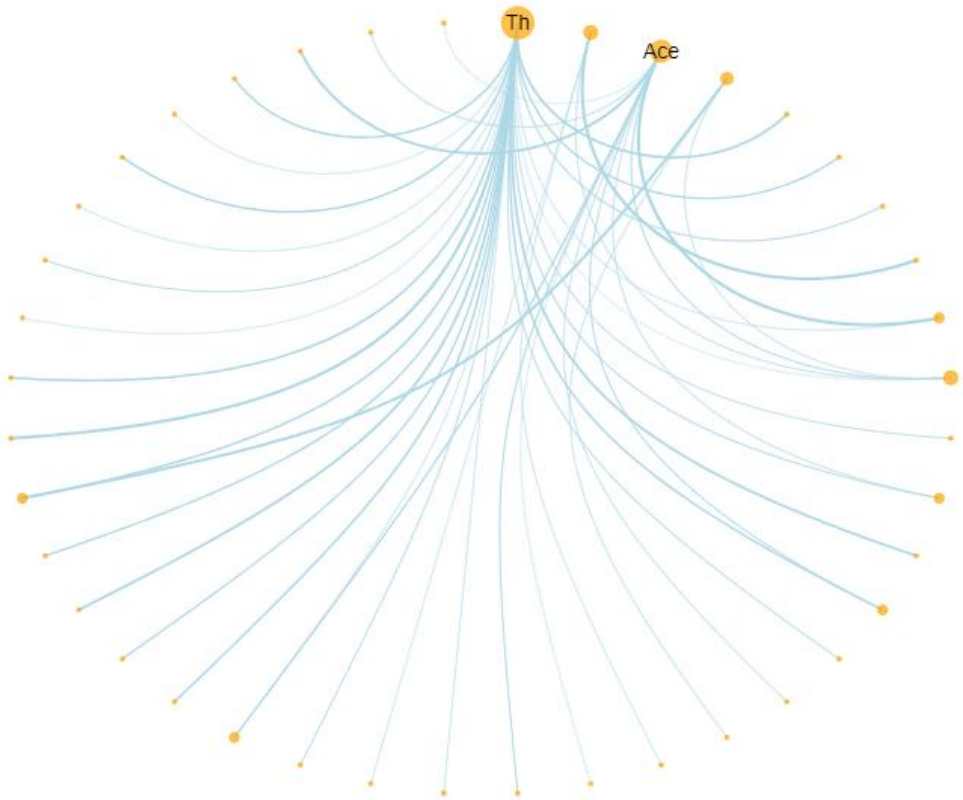
Example Commands

Run the script with default parameters:

```
Rscript ppi.R
```

Run the script with specified parameters:

```
Rscript ppi.R -i ppi_example.txt -a SYMBOL -b Homo_sapiens -c 400 -d linear -e 5
```



4. Hyperparameter optimization and performance evaluation

This section provides the steps for hyperparameter optimization to help users train better models. The following are the adjustment methods for each hyperparameter:

- kfold: Generally start by trying 3, 5, or 10; 5-fold cross-validation is commonly used.
- Dataset split ratio**:** Common train:validation:test set split ratios are 7:1.5:1.5 or 8:1:1. More training data is generally better.
- Batch size: Typically start by trying 32, 64, or 128. Smaller batch sizes usually provide better generalization but train slower. Larger batch sizes train faster but may overfit.
- Learning rate: Usually start training with 0.1, 0.01, 0.001, or 0.0001. Adjust the learning rate based on the Acc-loss curves. If the training loss curve does not decrease, try increasing the learning rate; if both training and validation loss curves plateau, it might be due to too small a learning rate or model convergence.
- Optimizer: Adam has strong adaptability and is the default optimizer; SGD requires more fine-tuned learning rate adjustment; RMSprop performs well in RNNs and certain tasks. We recommend trying Adam first, then SGD if higher accuracy is needed.
- Loss Function: CrossEntropy is the standard loss for multi-class problems; NLLLoss is used with LogSoftmax; Focal Loss is used to address class imbalance issues. Users can choose the appropriate loss function based on the class distribution of their dataset.
- Epochs: Number of training rounds. Users can initially set a relatively large value (e.g., 200) to observe the loss and accuracy during training and determine when overfitting or underfitting starts. Reduce epochs if overfitting occurs, and increase epochs if underfitting occurs. Rely on early stopping mechanisms to prevent overfitting.
- Early stopping patience: Start by trying 5, 8, or 10. If the validation loss fluctuates significantly, increase the patience value.

First optimize the learning rate and batch size, then fix the best learning rate and batch size to optimize the optimizer, and finally fine-tune other hyperparameters. By observing the training and validation loss and accuracy curves, one can diagnose the model's learning status. Ideally, both loss curves should decrease synchronously and

then converge to a low value, while both accuracy curves should rise synchronously and converge to a high value. If the validation loss begins to rise while the training loss continues to decrease, or if the validation accuracy is significantly lower than the training accuracy, it indicates overfitting; if both loss curves remain high, it suggests underfitting. When training the model, if the model overfits, increase the Early stopping patience; if the model underfits, adjust the learning rate; if training is unstable, reduce the learning rate and adjust the batch size; if convergence is slow, increase the learning rate and adjust the optimizer.

The test set's accuracy, precision, recall, and F1-score collectively evaluate the model's final generalization capability on unseen data. A high and balanced precision and recall (reflected by a high F1-score) is the hallmark of a robust model. Accuracy is valid for class-balanced datasets, but if class imbalance exists, more attention should be paid to the F1-score. The emphasis between precision and recall depends on the specific task: pursue high precision if false positives are unacceptable, and high recall if false negatives are unacceptable. An excellent model should perform comprehensively and well across these test metrics.