

# Package ‘flowMagic’

December 17, 2024

**Type** Package

**Title** Automated gating of flow cytometry data

**Version** 0.99

**Description** tool to perform the automated gating of bivariate flow cytometry data.

**License** Apache License 2.0 | file LICENSE + file NOTICE

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.1

**Imports** caret,  
ggplot2,  
stringr,  
parallel,  
randomForest,  
concaveman,  
sp,  
sf,  
stats,  
grDevices,  
doParallel,  
nnet,  
sm,  
pracma

**Suggests** flowCore,  
flowWorkspace,  
data.tree,  
visNetwork,  
klaR(<= 0.6-12),  
remotes,  
CytoML,

**R topics documented:**

add_labels_column . . . . .	3
assign_events_to_nearest_centroids . . . . .	3
check_polygons_intersection . . . . .	4
compute_gates . . . . .	4
csv_to_dens . . . . .	5
exports_plots . . . . .	6
extract_polygon_gates . . . . .	7
get_centroids . . . . .	7
get_classes_expr_df . . . . .	8
get_density_features . . . . .	8
get_density_scores . . . . .	9
get_distance_loc_vs_test . . . . .	9
get_dist_template . . . . .	10
get_hierarchy_all_pops . . . . .	11
get_hull_all_gates . . . . .	11
get_indices_cross_val . . . . .	12
get_local_train_sets . . . . .	13
get_pops_hierarchy_list . . . . .	13
get_pop_multiclass . . . . .	14
get_slot_hierarchy_list . . . . .	14
get_test_sets . . . . .	15
get_train_data . . . . .	16
get_weights_density_features . . . . .	17
import_gating_info . . . . .	17
import_reference_csv . . . . .	18
import_test_set_csv . . . . .	18
import_test_set_fcs . . . . .	19
magicPlot . . . . .	19
magicPred . . . . .	21
magicPred_all . . . . .	22
magicPred_hierarchy . . . . .	23
magicTrain . . . . .	23
magicTrain_dt . . . . .	24
magicTrain_hierarchy . . . . .	25
magicTrain_knn . . . . .	26
magicTrain_nb . . . . .	26
magicTrain_nnet . . . . .	27
magicTrain_rf . . . . .	28
name_pop_gating . . . . .	28
post_process_gates . . . . .	29
pre_process_manual_binary . . . . .	30
process_test_data . . . . .	30
range01 . . . . .	31
subsetting_binary_df . . . . .	31
update_label_association . . . . .	32

---

add_labels_column	<i>add_labels_column</i>
-------------------	--------------------------

---

**Description**

function to add label association column.

**Usage**

```
add_labels_column(df, labels_association)
```

**Arguments**

df	Dataframe.
labels_association	Vector of label association.

**Value**

Dataframe.

**Examples**

```
add_labels_column()
```

---

assign_events_to_nearest_centroids	<i>assign_events_to_nearest_centroids</i>
------------------------------------	---

---

**Description**

function to assign events to class with nearest centroid.

**Usage**

```
assign_events_to_nearest_centroids(gated_df, n_cores = 1,
  method_dist = "euclidean", thr_dist = 0.15, include_zero = F,
  remove_centroids = T)
```

**Arguments**

gated_df	dataframe with labels (third column).
n_cores	Number of cores. Default to 1.
method_dist	Distance method calculation. Default to euclidean.
thr_dist	Distance threshold for centroids calculation. Default to 0.15.
include_zero	Consider centroid of label 0. Default to False.
remove_centroids	Remove centroids too near each other based on thr_dist value.

**Value**

Dataframe.

**Examples**

assign\_events\_to\_nearest\_centroids()

---

check_polygons_intersection
<i>extract_polygon_gates</i>

---

**Description**

function to check polygons intersection.

**Usage**

check\_polygons\_intersection(list\_df\_hull)

**Arguments**

list\_df\_hull     List of polygons coordinates

**Value**

float

**Examples**

check\_polygons\_intersection()

---

compute_gates	<i>compute_gates</i>
---------------	----------------------

---

**Description**

function to assign events based on polygon gates.

**Usage**

compute\_gates(gated\_df, list\_final\_polygons\_coords, no\_classes = F)

**Arguments**

`gated_df`            dataframe with labels (third column).  
`list_final_polygons_coords`            List of dataframes containing polygon coordinates.  
`no_classes`            Generate third column of labels. Default to False.

**Value**

Dataframe.

**Examples**

```
compute_gates()
```

---

csv_to_dens	<i>csv_to_dens</i>
-------------	--------------------

---

**Description**

function to get density of events (with classes associated if present).

**Usage**

```
csv_to_dens(df, with_classes = T, n_coord = "df", normalize_data = T)
```

**Arguments**

`df`                    Dataframe of marker expression values.  
`with_classes`        Consider classes. Default to True.  
`n_coord`            Grid size. Default to df.  
`normalize_data`    If True, data is normalized to 0-1 range. Default to True.

**Value**

Dataframe.

**Examples**

```
csv_to_dens()
```

exports\_plots

*exports\_plots***Description**

function to generate plots (no hierarchy).

**Usage**

```
exports_plots(list_gated_data, path_output, n_cores = 1,
              type_plot = "dens", show_legend = T, x_lab = "x", y_lab = "y",
              size_title_x = 23, size_title_y = 23, aspect_ratio = NULL,
              w_val = 7, h_val = 7, size_axis_text = 25)
```

**Arguments**

list_gated_data	list of dataframes. Each dataframe has 3 columns: marker 1 values, marker 2 values and label column.
path_output	Path to the directory where to export the plots.
n_cores	Path to directory containing the expression data of the files analyzed.
type_plot	the user can choose between density (="dens) or label assignment visualization (="ML")
show_legend	If True it shows the legend for the label assignment visualization. Default to True.
x_lab	x-axis label.
y_lab	y-axis label.
size_title_x	Size x axis label.
size_title_y	Size y axis label.
aspect_ratio	Set aspect ratio= Default to NULL> If = 1, y and x axis ticks have same distance.
w_val	width value. Default to 7.
h_val	height value. Default to 7.
size_axis_text	Size of ticks labels.

**Examples**

```
exports_plots()
```

---

extract_polygon_gates	<i>extract_polygon_gates</i>
-----------------------	------------------------------

---

**Description**

function to extract the polygon gates objects based on the convex hull and classes.

**Usage**

```
extract_polygon_gates(gated_df, concavity_val = 1)
```

**Arguments**

gated_df	dataframe with labels (third column).
concavity_val	Concavity of polygons. Default to 1.

**Value**

List of dataframes.

**Examples**

```
extract_polygon_gates()
```

---

get_centroids	<i>get_centroids</i>
---------------	----------------------

---

**Description**

function to get centroids for each label

**Usage**

```
get_centroids(df, low_thr = 0.1, up_thr = 0.9, thr_dist = 0.15,  
              include_zero = F, remove_centroids = T)
```

**Arguments**

df	dataframe with labels (third column).
low_thr	Lower threshold for quantile calculation.
up_thr	Upper threshold for quantile calculation.
thr_dist	Distance threshold for centroids calculation. Default to 0.15.
include_zero	Consider centroid of label 0. Default to False.
remove_centroids	Remove centroids too near each other based on thr_dist value.

**Value**

Dataframe.

**Examples**

```
get_centroids()
```

---

```
get_classes_expr_df     get_classes_expr_df
```

---

**Description**

function to get classes of original expression df based on density df predictions.

**Usage**

```
get_classes_expr_df(dens_df, original_df)
```

**Arguments**

<code>dens_df</code>	Dataframe of density values.
<code>original_df</code>	Original dataframe.

**Value**

Dataframe.

**Examples**

```
get_classes_expr_df()
```

---

```
get_density_features     get_density_features
```

---

**Description**

function to get density features only given a bivarite density csv.

**Usage**

```
get_density_features(df_dens, min_height = 0.06)
```

**Arguments**

<code>df_dens</code>	Dataframe of density estimates for both markers.
<code>min_height</code>	Minimum height of the peaks to consider.



**Value**

Vector of numbers.

**Examples**

```
get_density_features()
```

---

```
get_density_scores    get_density_scores
```

---

**Description**

function to get scores for distance template calculation.

**Usage**

```
get_density_scores(df_template, df_test, select_density_features = NULL)
```

**Arguments**

df\_template      Dataframe of template markers expression values.  
df\_test            Dataframe of test markers expression values.  
select\_density\_features  
                    Select features to use. Default to NULL.

**Value**

Matrix of numbers.

**Examples**

```
get_density_scores()
```

---

```
get_distance_loc_vs_test  
                          get_distance_loc_vs_test
```

---

**Description**

function to compare the similiraty between current test data and local training set.

**Usage**

```
get_distance_loc_vs_test(test_df, loc_df, show_plot = "none",  
                          nboot = 50)
```

**Arguments**

<code>test_df</code>	Dataframe with bivariate marker expression.
<code>loc_df</code>	Dataframe with bivariate marker expression.
<code>show_plot</code>	show density comparison. Default to none.
<code>nboot</code>	Number of permutations. Default to 50.

**Value**

List of p-values.

**Examples**

```
get_distance_loc_vs_test()
```

---

<code>get_dist_template</code>	<i>get_dist_template</i>
--------------------------------	--------------------------

---

**Description**

function to get distance between template and test data.

**Usage**

```
get_dist_template(matrix_scores, dist_method = "euclidean")
```

**Arguments**

<code>matrix_scores</code>	Matrix of density features generated by <code>get_density_scores</code> function.
<code>dist_method</code>	Type of distance method calculation.

**Value**

Number.

**Examples**

```
get_dist_template()
```

---

```
get_hierarchy_all_pops
    get_hierarchy_all_pops
```

---

### Description

function to get the hierarchy of all pops from the sample manually gated (the input gating hierarchy). Based on its output the function magicTrain will perform the training step using the sample manually gated (local training set) and the project discovery data (global training set).

### Usage

```
get_hierarchy_all_pops(gh, export_visnet = F, path.output = "None")
```

### Arguments

gh	GatingHierarchy.
export_visnet	If true, it export visnetwork object.
path.output	Path to save visnetwork object. Default to None.

### Value

List of Dataframes.

### Examples

```
get_hierarchy_all_pops()
```

---

```
get_hull_all_gates    get_hull_all_gates
```

---

### Description

function to get the convex hull of all gates.

### Usage

```
get_hull_all_gates(gated_df, concavity_val = 1)
```

### Arguments

gated_df	dataframe with labels (third column).
concavity_val	Values of concavity. Default to 1.

**Value**

List of dataframes.

**Examples**

```
get_hull_all_gates()
```

---

```
get_indices_cross_val  get_indices_cross_val
```

---

**Description**

function to get indices for cross val.

**Usage**

```
get_indices_cross_val(df_train, n_cores = 1, train_inds = "plot_num",
                      val_inds = "none", n_train_plots = 5, n_folds = 5, seed = 40,
                      n_val_plots = 5)
```

**Arguments**

<code>df_train</code>	Dataframe of training features generated by the <code>get_train_data</code> function.
<code>n_cores</code>	Number of cores to use. Default to 1.
<code>train_inds</code>	Type of method to extract training indices: <code>plot_num</code> , <code>rand_set_num</code> , <code>rand_set_n_gates_info</code> .
<code>val_inds</code>	Type of method to extract validation indices: <code>plot_num</code> , <code>rand_set_num</code> , <code>rand_set_n_gates_info</code> .
<code>n_train_plots</code>	Number of training plots to include in training data for each iteration.
<code>n_folds</code>	Number of training iterations.
<code>seed</code>	Seed to randomly extract data.
<code>n_val_plots</code>	Number of training plots to include in validation data for each iteration.

**Value**

List of integers.

**Examples**

```
get_indices_cross_val()
```

---

get_local_train_sets	<i>get_local_train_sets</i>
----------------------	-----------------------------

---

**Description**

Based on the hierarchy calculated using the `get_hierarchy` function, we generate the local training sets gated by the biologists.

**Usage**

```
get_local_train_sets(gh, hierarchical_tree, info_hierarchy)
```

**Arguments**

gh	GatingHierarchy.
hierarchical_tree	Dataframe of hierarchy information generated by the <code>get_hierarchy</code> function.
info_hierarchy	List of hierarchy information generated by the <code>get_hierarchy</code> function.

**Value**

List of Dataframes.

**Examples**

```
get_local_train_sets()
```

---

get_pops_hierarchy_list
<i>get_pops_hierarchy_list</i>

---

**Description**

function to get info from a list of hierarchical dataset (it reports all the pops for each level in a vector).

**Usage**

```
get_pops_hierarchy_list(hierarchical_list)
```

**Arguments**

hierarchical_list	list of populations names for each level.
-------------------	---

**Value**

Vector of characters.

**Examples**

```
get_pops_hierarchy_list()
```

---

<code>get_pop_multiclass</code>	<i>get_pop_multiclass</i>
---------------------------------	---------------------------

---

**Description**

function to get the children of the selected population, useful for multiclass classification. The output is only the pops with same dimensions

**Usage**

```
get_pop_multiclass(gh, pop)
```

**Arguments**

<code>gh</code>	GatingHierarchy.
<code>pop</code>	Name of population to get events assignments.

**Value**

Vector of characters.

**Examples**

```
get_pop_multiclass()
```

---

<code>get_slot_hierarchy_list</code>	<i>get_slot_hierarchy_list</i>
--------------------------------------	--------------------------------

---

**Description**

function to access a slot of results from the hierarchy list based on the pop selected.

**Usage**

```
get_slot_hierarchy_list(hierarchical_list, pop_selected)
```

**Arguments**

hierarchical\_list  
list of populations names for each level.

pop\_selected    Population name.

**Value**

Element of a list.

**Examples**

```
get_slot_hierarchy_list()
```

---

get_test_sets	<i>get_test_sets</i>
---------------	----------------------

---

**Description**

Function to obtain the data ready to be gated (validation set or test set). It takes as input a flowSet and generates a list of dataframes with a ML structure for each ungated fcs file. Each dataframe is the expression matrix of the root.

**Usage**

```
get_test_sets(fs, gh)
```

**Arguments**

fs                    flowSet to gate.

gh                    Gating hierarchy.

**Value**

List of dataframes.

**Examples**

```
get_test_sets()
```

---

get_train_data	<i>get_train_data</i>
----------------	-----------------------

---

### Description

function to import training data based on paths to files.

### Usage

```
get_train_data(paths_file = NULL, df_paths = NULL, n_cores = 1,
               prop_down = NULL, remove_class = NULL, n_points_per_plot = NULL,
               normalize_data = T, vec_col = NULL)
```

### Arguments

paths_file	Vector of paths. Each path points toward a single csv file containin training info (labels and bivariate expression). paths_file can be also directly the list of dataframes containing labels and bivariate expression.
df_paths	Dataframe containing the paths of file to read. The paths to data must be in the first column. The associated paths to classes are in second column.
n_cores	Number of cores. Default to 1.
prop_down	Proportion of events (downsampling). Default to NULL (downsampling using number of points).
remove_class	Vector of classes to ignore. Default to NULL.
n_points_per_plot	Number of points for downsampling.
normalize_data	If True, data is normalized to 0-1 range. Default to True.
vec_col	vector of columns names if the input dataframes have more than 3 columns. The third column name must always refer to the column with the gate label of each event. Default to NULL.

### Value

Dataframe.

### Examples

```
get_train_data()
```



---

```
get_weights_density_features
    get_weights_density_features
```

---

**Description**

function to calculate final score based on density features.

**Usage**

```
get_weights_density_features(df_scores)
```

**Arguments**

df\_scores            Dataframe of distance scores.

**Value**

Vector of numbers.

**Examples**

```
get_weights_density_features()
```

---

```
import_gating_info      import_sample_gated
```

---

**Description**

convert a flowWorkspace or a GatingML file of the train data folder into a gated gh (gh\_train)

**Usage**

```
import_gating_info(path, type = "gs", group_wsp = NULL)
```

**Arguments**

path                    path to gs or GatingML object.  
type                    type of object.  
group\_wsp              Group of wsp to import.

**Value**

GatingSet object

**Examples**

```
import_sample_gated()
```

---

```
import_reference_csv    import_reference_csv
```

---

**Description**

function to import plain gold standards data (no hierarchy)

**Usage**

```
import_reference_csv(path_results, n_cores = 1)
```

**Arguments**

`path_results`    path to directory containing the csv files to read (with third column of labels).  
`n_cores`        Number of cores to use. Default to 1.

**Value**

list of dataframes

**Examples**

```
import_reference_csv()
```

---

```
import_test_set_csv    import_test_set_csv
```

---

**Description**

function to import test set in csv format.

**Usage**

```
import_test_set_csv(path_data, n_cores = 1, xy_col = T)
```

**Arguments**

`path_data`        path to directory containing csv files to read (third column is ignored).  
`n_cores`        Number cores. Default to 1.  
`xy_col`        Colnames equal to x and y. Default to True.

**Value**

List of dataframes.

**Examples**

```
import_test_set_csv()
```

---

import_test_set_fcs	<i>import_test_set</i>
---------------------	------------------------

---

### Description

read the ungated fcs files into a flowSet. The ungated fcs are assumed to be already cleaned,compensated,and transformed.

### Usage

```
import_test_set_fcs(path, n_samples = "All", ref_f_n = 1)
```

### Arguments

path	path of directory containig the fcs files.
n_samples	Number of samples. Default to All.
ref_f_n	Set reference flowFrame to match channel names. Default to 1(first flowFrame).

### Value

flowSet.

### Examples

```
import_test_set()
```

---

magicPlot	<i>magicPlot</i>
-----------	------------------

---

### Description

function to generate the scatter plot with colored density of the events.

### Usage

```
magicPlot(df, type = "dens", polygons_coords_list = NULL,
  show_legend = T, size_axis_text = 18, size_title_x = 20,
  size_title_y = 20, treat_0_as_gate = F, x_lab = "x", y_lab = "y",
  gates_to_plot = NULL, apply_manual_scale = T, hull_only = F,
  size_points = 1, concavity_val = 5, aspect_ratio = NULL,
  x_lim1 = NULL, x_lim2 = NULL, y_lim1 = NULL, y_lim2 = NULL)
```

**Arguments**

<code>df</code>	Dataframe of bivariate markers expression (with labels if gates to plot).
<code>type</code>	Type of plot to generated. "dens"=bivariate density plot. "ML"=events assignments plot
<code>polygons_coords_list</code>	list of gates coordinates. Needed if labels not included in df. Default to NULL.
<code>show_legend</code>	Show legend if type="ML". Default to True.
<code>size_axis_text</code>	Size of axis ticks labels. Default to 18.
<code>size_title_x</code>	Size of x axis label title.
<code>size_title_y</code>	Size of y axis label title.
<code>treat_0_as_gate</code>	Treat 0 label as gate. Default to False (0 label is background)
<code>x_lab</code>	Label of x axis.
<code>y_lab</code>	Label off y axis.
<code>gates_to_plot</code>	Select labels to plot.
<code>apply_manual_scale</code>	Apply predefined scale of colors. Default to True.
<code>size_points</code>	Size of points in scatter plot.
<code>concavity_val</code>	Concavity value. Default to 5.
<code>aspect_ratio</code>	Aspect ratio value. If = 1, y and x axis have equal distance between the ticks labels. Default to NULL.
<code>x_lim1</code>	Minimum limit x axis. Default to NULL.
<code>x_lim2</code>	Max limit x axis. Default to NULL.
<code>y_lim1</code>	Minimum limit y axis. Default to NULL.
<code>y_lim2</code>	Max limit y axis. Default to NULL.

**Value**

ggplot.

**Examples**

```
magicPlot()
```

---

magicPred

*magicPred*


---

## Description

function to predict on plain test data (no hierarchy)

## Usage

```
magicPred(test_data, magic_model = NULL, magic_model_n_gates = NULL,
          ref_model_info = NULL, n_cores = 1, ref_data_train = NULL,
          prop_down = NULL, thr_dist = 0.05, n_points_per_plot = NULL,
          normalize_data = T, include_zero_val = T)
```

## Arguments

test_data	Dataframe of test data to gate. It has only the two columns of marker expression.
magic_model	Global trained model to predict gates. It can be a single model or list of named models (each model trained on selected number of gates).
magic_model_n_gates	Global trained model to predict number of gates. If different from NULL, magic_model is expected to be a list of models to predict certain gates (e.g., 5 models for 2,3,4,5 or 6 gates).
ref_model_info	Template model to predic gates.
n_cores	Number of cores to use. Default to 1.
ref_data_train	Template data used to generate ref_model_info. Needed to calculate target-template distance.
prop_down	Proportion for downsampling. Default to NULL (automatic downsampling using n_points_per_plot).
n_points_per_plot	Number of points to consider for downsampling. Default to 500.
normalize_data	If True, data is normalized to 0-1 range. Default to True.
include_zero_val	considering events labeled as 0 as an additional gate when there is only one gate. Default to True.

## Value

List of Dataframes.

## Examples

```
magicPred()
```

---

magicPred\_all

*magicPred\_all*


---

### Description

function to predict on plain test data (no hierarchy)

### Usage

```
magicPred_all(list_test_data, magic_model = NULL,
              ref_model_info = NULL, magic_model_n_gates = NULL,
              ref_data_train = NULL, prop_down = NULL, n_points_per_plot = NULL,
              thr_dist = 0.05, n_cores = 1, normalize_data = T,
              include_zero_val = T, n_cores_all = 1)
```

### Arguments

**list\_test\_data** List of unlabeled dataframes. It has only the two columns of marker expression.

**magic\_model** Global trained model to predict gates. It can be a single model or list of named models (each model trained on selected number of gates).

**ref\_model\_info** Template model to predic gates.

**magic\_model\_n\_gates** Global trained model to predict number of gates. If different from NULL, **magic\_model** is expected to be a list of models to predict certain gates (e.g., 5 models for 2,3,4,5 or 6 gates).

**ref\_data\_train** Template data used to generate **ref\_model\_info**. Needed to calculate target-template distance.

**n\_points\_per\_plot** Number of points to consider for downsampling. Default to 500.

**n\_cores** Number of cores to use to process one sample. Default to 1.

**normalize\_data** If True, data is normalized to 0-1 range. Default to True.

**include\_zero\_val** considering events labeled as 0 as an additional gate when there is only one gate. Default to True.

**n\_cores\_all** Number of cores to use across all samples. Default to 1.

### Value

List of Dataframes.

### Examples

```
magicPred_all()
```

---

magicPred_hierarchy	<i>magicPred_hierarchy</i>
---------------------	----------------------------

---

**Description**

function to predict the gates on the ungated .fcs samples.

**Usage**

```
magicPred_hierarchy(list_test_sets, list_models_local, df_tree,
  n_cores = 1)
```

**Arguments**

<code>list_test_sets</code>	contains the list of root dataframe for each ungated fcs file imported.
<code>list_models_local</code>	contains the optimized local models pre-generated using the <code>magicTrain_local</code> function.
<code>df_tree</code>	contains the info related to the populations hierarchy.
<code>n_cores</code>	Number of cores to use. Default to 1.

**Value**

List of Dataframes.

**Examples**

```
magicPred_hierarchy()
```

---

magicTrain	<i>magicTrain</i>
------------	-------------------

---

**Description**

function to generate one training model based on a list of training sets (no hierarchy).

**Usage**

```
magicTrain(df_train, n_cores = 1, train_model = "rf", k_cv = 10,
  list_index_train = NULL, list_index_val = NULL, n_tree = 10,
  tune_lenght = 3, size_nnet_units = 100, decay_nnet = 0.1,
  method_control = "oob", type_y = "classes", seed_n = 40)
```

**Arguments**

df_train	training dataframe generated by the get_train_data function.
n_cores	Number of cores to use. Default to 1.
train_model	Type of training model. Default to rf.
k_cv	Number of k for cross-validation (if method control=cv)
list_index_train	List of vector of indices to use in training for each fold.
list_index_val	List of vector of indices to use as held out data for each fold.
n_tree	Number of trees for random forest.
tune_lenght	Number of parameters values trained during cross-validation.
size_nnet_units	Number of units in hidden layer (if train_model=nnet).
decay_nnet	Decay parameter value for nnet model.
method_control	Type of training control: oob or cv. Default to oob.
type_y	Type of response variable: classes (train to predict gates boundaries) or n_gates_info(train to predict number of gates).
seed_n	Set seed. Default to 40.

**Value**

model object.

**Examples**

```
magicTrain()
```

---

magicTrain\_dt

*magicTrain\_dt*


---

**Description**

function to generate a random forest training model.

**Usage**

```
magicTrain_dt(Xtrain, Ytrain, k_cv = 10, list_index_train = NULL,
  list_index_val = NULL, tune_lenght = 5)
```



**Arguments**

Xtrain	Dataframe of training features.
Ytrain	Dataframe of labels (one column).
k_cv	Number of k for cross-validation.
list_index_train	List of vector of indices to use in training for each fold.
list_index_val	List of vector of indices to use as held out data for each fold.
tune_lenght	Number of hyper parameters to test. Default to 5.

**Value**

model object.

**Examples**

```
magicTrain_dt()
```

---

magicTrain\_hierarchy    *magicTrain\_local*

---

**Description**

function to generate the local training models using the list of hierarchical training set.

**Usage**

```
magicTrain_hierarchy(list_train_sets, n_tree = 10, train_model = "rf",
  method_control = "oob", n_cores = 1)
```

**Arguments**

list_train_sets	List of labeled dataframe to train generated by the get_local_train function.
n_tree	Number of tree for random forest model.
train_model	Type of training model. Default to random forest ("rf").
method_control	Cross-validation method. Default to out-of-the-bag method (oob).
n_cores	Number of cores to use. Default to 1.

**Value**

List of models objects.

**Examples**

```
magicTrain_local()
```

---

magicTrain_knn	<i>magicTrain_knn</i>
----------------	-----------------------

---

**Description**

function to generate a random forest training model.

**Usage**

```
magicTrain_knn(Xtrain, Ytrain, k_cv = 10, list_index_train = NULL,
               list_index_val = NULL, tune_lenght = 5)
```

**Arguments**

Xtrain	Dataframe of training features.
Ytrain	Dataframe of labels (one column).
k_cv	Number of k for cross-validation.
list_index_train	List of vector of indices to use in training for each fold.
list_index_val	List of vector of indices to use as held out data for each fold.
tune_lenght	Number of hyper parameters to test. Default to 5.

**Value**

model object.

**Examples**

```
magicTrain_knn()
```

---

magicTrain_nb	<i>magicTrain_nb</i>
---------------	----------------------

---

**Description**

function to generate a random forest training model.

**Usage**

```
magicTrain_nb(Xtrain, Ytrain, k_cv = 10, list_index_train = NULL,
               list_index_val = NULL, tune_lenght = 5)
```

**Arguments**

Xtrain	Dataframe of training features.
Ytrain	Dataframe of labels (one column).
k_cv	Number of k for cross-validation.
list_index_train	List of vector of indices to use in training for each fold.
list_index_val	List of vector of indices to use as held out data for each fold.
tune_lenght	Number of hyper parameters to test. Default to 5.

**Value**

model object.

**Examples**

```
magicTrain_nb()
```

---

magicTrain_nnet	<i>magicTrain_nnet</i>
-----------------	------------------------

---

**Description**

function to generate a neural net training model.

**Usage**

```
magicTrain_nnet(Xtrain, Ytrain, k_cv = 10, list_index_train = NULL,
  list_index_val = NULL, size = 100, decay = 0.1, tune_lenght = 5)
```

**Arguments**

Xtrain	Dataframe of training features.
Ytrain	Dataframe of labels (one column).
k_cv	Number of k for cross-validation.
list_index_train	List of vector of indices to use in training for each fold.
list_index_val	List of vector of indices to use as held out data for each fold.
size	Number of units in hidden layer (if train_model=nnet).
decay	Decay parameter value for nnet model.

**Value**

model object.

**Examples**

```
magicTrain_nnet()
```

---

magicTrain_rf	<i>magicTrain_rf</i>
---------------	----------------------

---

**Description**

function to generate a random forest training model.

**Usage**

```
magicTrain_rf(Xtrain, Ytrain, list_index_train = NULL,
              list_index_val = NULL, n_tree = 10, method_control = "oob",
              k_cv = 10)
```

**Arguments**

- Xtrain            Dataframe of training features.
- Ytrain            Dataframe of labels (one column).
- list\_index\_train            List of vector of indices to use in training for each fold.
- list\_index\_val   List of vector of indices to use as held out data for each fold.
- n\_tree            Number of trees for random forest.
- method\_control   Type of training control: oob or cv. Default to oob.
- k\_cv              Number of k for cross-validation (if method control=cv).

**Value**

model object.

**Examples**

```
magicTrain_rf()
```

---

name_pop_gating	<i>name_pop_gating</i>
-----------------	------------------------

---

**Description**

function to get the name of all the pops of the gating hierarchy.

**Usage**

```
name_pop_gating(gh)
```

**Arguments**

gh                      GatingHierarchy.

**Value**

Vector of characters.

**Examples**

```
name_pop_gating()
```

---

post_process_gates	<i>post_process_gates</i>
--------------------	---------------------------

---

**Description**

function to post process the events after model prediction.

**Usage**

```
post_process_gates(gated_df, n_cores = 1, thr_dist = 0.15,
  include_zero = F, remove_centroids = T, type = "dist",
  concavity_val = 5, normalize_data = T)
```

**Arguments**

gated\_df              dataframe with labels (third column).  
n\_cores                Number of cores. Default to 1.  
thr\_dist               Distance threshold for centroids calculation. Default to 0.15.  
include\_zero          Consider centroid of label 0. Default to False.  
remove\_centroids      Remove centroids too near each other based on thr\_dist value.  
type                    Type of post-processing.  
concavity\_val          Concavity of polygons for the "polygon" type of post-processing

**Value**

Dataframe.

**Examples**

```
post_process_gates()
```

---

```
pre_process_manual_binary
      pre_process_manual_binary
```

---

### Description

In other words it generates the dataset that indicates what cells belong to the selected pop (1) 0 otherwise. The dataset is always the dataset of the Root pop. gh and pop are mandatory arguments, dim and mode have a default value.

### Usage

```
pre_process_manual_binary(gh, pop)
```

### Arguments

gh	GatingHierarchy.
pop	Name of the population to get events assignments.

### Value

Dataframe.

### Examples

```
pre_process_manual_binary()
```

---

```
process_test_data      process_test_data
```

---

### Description

function to get test data correctly formatted.

### Usage

```
process_test_data(test_data, prop_down = NULL, n_points_per_plot = 500,
  normalize_data = T)
```

### Arguments

test_data	Dataframe of bivariate markers expression.
prop_down	Proportion of events (downsampling). Default to NULL (downsampling using number of points).
n_points_per_plot	Number of points for downsampling.
normalize_data	If True, data is normalized to 0-1 range. Default to True.

**Value**

Dataframe.

**Examples**

```
process_test_data()
```

---

range01
---------

---

<i>range01</i>
----------------

---

**Description**

function to put data in range 0-1.

**Usage**

```
range01(x)
```

**Arguments**

x	Vector of numbers to scale.
---	-----------------------------

**Value**

Vector of numbers.

**Examples**

```
range01()
```

---

subsetting_binary_df
----------------------

---

<i>subsetting_binary_df</i>
-----------------------------

---

**Description**

function to get binary dataset with gate assignation based on the mother population of the selected pop (instead of the root).

**Usage**

```
subsetting_binary_df(gh, binary_df)
```

**Arguments**

gh	GatingHierarchy.
binary_df	Dataframe generated by the pre_process_manual_binary function.

**Value**

Dataframe.

**Examples**

```
subsetting_binary_df()
```

---

```
update_label_association
```

```
update_label_association
```

---

**Description**

function to update label association.

**Usage**

```
update_label_association(df)
```

**Arguments**

df                      Dataframe.

**Value**

Vector.

**Examples**

```
update_label_association()
```



# Index

`add_labels_column`, 3  
`assign_events_to_nearest_centroids`, 3  
  
`check_polygons_intersection`, 4  
`compute_gates`, 4  
`csv_to_dens`, 5  
  
`exports_plots`, 6  
`extract_polygon_gates`, 7  
  
`get_centroids`, 7  
`get_classes_expr_df`, 8  
`get_density_features`, 8  
`get_density_scores`, 9  
`get_dist_template`, 10  
`get_distance_loc_vs_test`, 9  
`get_hierarchy_all_pops`, 11  
`get_hull_all_gates`, 11  
`get_indices_cross_val`, 12  
`get_local_train_sets`, 13  
`get_pop_multiclass`, 14  
`get_pops_hierarchy_list`, 13  
`get_slot_hierarchy_list`, 14  
`get_test_sets`, 15  
`get_train_data`, 16  
`get_weights_density_features`, 17  
  
`import_gating_info`, 17  
`import_reference_csv`, 18  
`import_test_set_csv`, 18  
`import_test_set_fcs`, 19  
  
`magicPlot`, 19  
`magicPred`, 21  
`magicPred_all`, 22  
`magicPred_hierarchy`, 23  
`magicTrain`, 23  
`magicTrain_dt`, 24  
`magicTrain_hierarchy`, 25  
`magicTrain_knn`, 26  
`magicTrain_nb`, 26  
  
`magicTrain_nnet`, 27  
`magicTrain_rf`, 28  
  
`name_pop_gating`, 28  
  
`post_process_gates`, 29  
`pre_process_manual_binary`, 30  
`process_test_data`, 30  
  
`range01`, 31  
  
`subsetting_binary_df`, 31  
  
`update_label_association`, 32