

flowMagic: automated gating of bivariate flow cytometry data

Sebastiano Montante

January 29, 2026

Abstract

This pdf describes the usage of the main flowMagic functions and reports examples of typical flowMagic scripts. Each function usually contains many parameters, only few of them are mentioned in this document. A detailed documentation of each parameter for each function can be found on the flowMagic_manual.pdf document in the pkg_manual github directory.

1 Installation

The easiest way to install flowMagic is directly from github:

```
install_github("semontante/flowMagic",ref="main")
```

The user can also download the package locally and install it from the package folder:

```
install.packages("path/to/flowMagic.tar.gz",repos=NULL,type="source")
```

The following libraries are required:

```
library(sp)
library(stringr)
library(ggplot2)
library(parallel)
library(doParallel)
library(randomForest)
library(caret)
library(concaveman)
library(sm)
library(pracma)
library(sf)
library(stats)
library(grDevices)
library(flowMagic)
```

Note: flowMagic was tested on R 3.5.2 with the sf package version 0.7.2. It is highly recommended to use the provided docker container where all the required packages are installed.

2 Input

The user can set up two types of workflows when using flowMagic:

1. A workflow that starts from the CSV files (extracted from gated flow cytometry data).
2. A workflow that starts from the popular FCS files.

2.1 CSV files

There are 2 types of input: ungated data to analyze and the trained model to use for gating. The ungated data is within a directory containing the bivariate marker expression of the images under analysis. In particular, the marker expression must be reported in a csv file whose first and second column report the expression of, respectively, the first and second marker. Each row refers to the expression of a single event/cell. Here's an example of a correctly formatted csv file:

"PE-CF594-A"	"FITC-A"
2.985	3.137
2.433	2.752
2.942	2.914
2.874	3.121
1.805	2.119
1.925	2.712
0.871	2.56
2.782	2.836
2.645	2.265

The trained model can be either a templates model or a generalized model. The template model is needed to automatically gate the data based on the patterns defined by the user. The training data to generate this model is extracted by the user from the data to analyze containing the same combination of markers. The generalized model is trained on a large FCM dataset containing different combinations of markers extracted from different projects unrelated to the dataset under analysis. The flowMagic package includes a generalized model in the inst/data folder trained on the Project Discovery data. The users can train their own generalized model if they manage to generate a large dataset of FCM data with different combinations of markers. Both the template model and the generalized model require the training data in csv format with 3 columns for each csv file to train. Each csv file represents the bivariate marker expression (first and second column) associated with the gates (the third column, which is also classes column). Each numerical label indicates a different gate. Note that the 0 label always indicates the background events. In other words, they are events without gates.

PE-CF594-A	FITC-A	Classes
2.985	3.137	1
2.433	2.752	2
2.942	2.914	2
2.874	3.121	1
1.805	2.119	0
1.925	2.712	0
0.871	2.56	1
2.782	2.836	1
2.645	2.265	1

2.2 FCS files

It is also possible to directly analyze the popular FCS files. Later we will show an example of a workflow starting from the raw FCS files to the final gated data. The flowMagic package is designed to work with the flowCore and flowWorkspace R packages.

3 Visualization options

It is possible to visualize the FCM data using the visualization framework of the flowMagic package. The flowMagic visualization possibilities are numerous and some of them will be described along the workflows script in this document. However, the manual pdf file contains the full list of visualization functions integrated in the flowMagic package, thus it is recommended that users look at the full functions manual available at the flowMagic github page (pk manual page). The visualization framework is structured around a core function: magicPlot(). This function represents the foundation of all visualizations generated by the flowMagic package.

The magicPlot() function can be used to plot the scatter plot of the gated plot of interest. It is possible to visualize the gates assignment on a standard scatter plot (with the events colored based on their gates) or it is possible to visualize the polygons on a bivariate density scatter plot.

```
# Visualization of data exported from CSV files
magicPlot(df = df_temp,type = "ML") # gates assignment visualization

magicPlot(df = df_temp,type = "dens") # Bivariate density plot with polygons visualization.

# Visualization of FCS data.
flowMagic::magicPlot_fs(fs = fs,sample_id = 1,channel_x = "FSC.A",channel_y = "FSC.H")

# Export ungated data of all samples for visualization
flowMagic::export_raw_gs_plots(gs = gs,node_name = "root",
                              channel_x = "FSC.A",channel_y = "FSC.H",
                              path_output = "path/to/dyr")
```

It is highly suggested to use the help() function provided by base R to check all the possible visualization options.

```
help("magicPlot") # check all possible visualization options
help("export_raw_gs_plots") # check all possible export options
```

4 flowMagic workflow analyzing CSV files

4.1 Automated gating using the template model

First, it is necessary to import the ungated data using the `import_test_set_csv` function. The user needs to indicate the path to a directory containing all unlabeled csv files under analysis. If there are more than two columns, the function will import only the first two columns. The user can also parallelize the importing process using more cores.

```
list_test_data<-import_test_set_csv(path_data = "path/to/data", n_cores=8)
```

`list_test_data` is a list in which each element is a dataframe of two columns containing the bivariate marker expression of each CSV file. To generate the template model it is necessary to import the reference data to use for training. The user needs to provide the path to the directory containing the labeled csv files. The CSV files need to contain 3 columns with the third column reporting the labels of each event.

```
list_data_ref<-import_reference_csv(path_results = "path/to/data",n_cores = 8)
```

`list_data_ref` is also a list of dataframes. This list is the input of the pre-processing function needed to prepare the data for training. The `get_train_data` function generates the correctly formatted training set from the raw reference data, extracting the density features needed for the training. The function includes several parameters. For example, the user can choose the number of cores to speed up the pre-processing or they can choose to perform a downsampling of the data. By default, the function considers 90% of the input data. The data is also normalized by default, the user can also choose to disable normalization. If there are bivariate plots with extreme outliers, like sparse events in an angle of the plot, disabling normalization may improve accuracy.

```
# normalized data, no downsampling
ref_train<-get_train_data(paths_file = list_data_ref,n_cores = 8)

# normalized data, yes downsampling
ref_train<-get_train_data(paths_file = list_data_ref,prop_down = 0.90, n_cores = 8) # consider only 90% of the
events for each plot.

ref_train<-get_train_data(paths_file = list_data_ref,n_points_per_plot = 500, n_cores = 8) # consider only 500
points for each plot.

# no normalized data, no downsampling
ref_train<-get_train_data(paths_file = list_data_ref,n_cores = 8, normalize_data=F)
```

Then, the user needs to select the indices of the train set and validation set for the cross-validation method to perform during training. There are multiple possible methods, see the appropriate function documentation for the complete list of the methods. Below it is shown an example that performs the leave-out-out cross-validation.

```
list_inds_cross_val<-get_indices_cross_val(df_train = ref_train,
train_inds = "leave_one_out",val_inds = "leave_one_out")
```

After this, the training can begin. The user can choose the model to use for training. Each model has different parameters with their own default values that the users can change. The random forest with the default number of trees (`n_trees=10`) is used by default:

```
ref_model_info<-magicTrain(df_train = ref_train,train_model = "rf",
list_index_train = list_inds_cross_val$inds_train,list_index_val = list_inds_cross_val$inds_val)
```

In case of one template or 2 templates, it is suggested to use the out-of-bag cross validation which is the default cross-validation method in these cases. Note that the user does not need to input the cross-validation indices when using the out-of-bag method.

```
ref_model_info<-magicTrain(df_train = ref_train,train_model = "rf",
method_control="oob")
```

Finally, the prediction step can be performed. The user needs to provide the unlabeled data imported at the beginning and the trained model. If the user also provides the pre-processed data used for training (the previous `ref_train` variable), the prediction function can also calculate the template-target distance for further analysis.

```
list_dfs_pred<-magicPred_all(list_test_data = list_test_data, ref_model_info = ref_model_info,ref_data_train = ref_train)
```

The `list_dfs_pred` variable is a nested list. Each element of the list contains the prediction information related to one plot. Each prediction information consists of a list of several dataframes and other outputs referring to different steps of the prediction process for one plot. The most important dataframe is the dataframe reporting the predicted labels associated with each events of the input original data.

```
# Selecting the final dataframe of the first gated plot.
# The third column contains the predicted labels.
df_temp<-list_dfs_pred[[1]]$final_df
```

See the appropriate function documentation for details on the other outputs. If the user provided the pre-processed training set used for training, the `vec.dist` slot will contain the vector of target-template distances for each plot used as template. The other dataframes of each nested element refer to the gating of the downsampled data (in case downsampling is applied during the prediction step) or the normalized data. No downsampling is applied by default when using the template model. The downsampling is applied by default only when using the generalized model.

4.2 Automated gating using the generalized model

To apply the generalized model, the user can use the same prediction function described in the previous section. The only difference is related to the arguments used. There are two models to use in this case. Model A predicts the number of gates in the plot, while Model B predicts the gates boundaries based on the predicted number of gates. There is a different Model B for each possible number of gates (e.g., Model B.2 predicts the two gates boundaries, Model B.3 predicts 3 gates boundaries). The `magic_model` argument requires the list of Model B for each number of gates, while the `magic_model_n_gates` requires Model A. Based on the value predicted by Model A, the appropriate Model B.X is used from the list of Model B. The models must be downloaded from the Federated Research Data Repository (FRDR) at the link: <https://doi.org/10.20383/103.01352>

```
out_pred<-magicPred_all(list_test_data = list_test_data,magic_model = list_magic_models, magic_model_n_gates = random_forest_model_pred_n_gates_index, n_cores = 8)
```

The users can also generate their own generalized model using the `flowMagic` training function as described in the next section. It is also possible to force the function to predict a pre-defined number of gates. It is sufficient to replace the number of gates model with an integer indicating the number of gates. The appropriate Model B will be selected from the list of Model B provided.

```
out_pred<-magicPred_all(list_test_data = list_test_data,magic_model = list_magic_models,magic_model_n_gates = 3,n_cores = 8) # to predict boundaries associated to 3 gates.
out_pred<-magicPred_all(test_data = list_test_data,magic_model = list_magic_models,magic_model_n_gates = 4,n_cores = 8) # to predict boundaries associated to 4 gates.
```

Finally, it is also possible to provide a single model predicting directly the gate boundaries.

```
out_pred<-magicPred_all(list_test_data = list_test_data,magic_model = single_model,magic_model_n_gates = NULL,n_cores = 8)
```

By default, when applying the generalized model, the data are down-sampled to 500 points to speed up execution. The polygons calculated in the down-sampled data will be projected to the original data to get the true number of events for each gate. See the function documentation for details of each argument.

IMPORTANT: the default down-sampling to 500 points may lower accuracy in certain populations (especially populations with many events). It is suggested to reduce the downsampling to increase the accuracy, for example, considering 10,000-15,000 points instead of the default 500 points.

```
out_pred<-magicPred_all(list_test_data = list_test_data,magic_model = list_magic_models,magic_model_n_gates = 4,n_cores = 8) # dowsampling to 15000 points instead of the default 500 points
```

4.3 Example of full scripts using either template or generalized model on CSV files

Example of correct script using the template model.

```

# load libraries
library(sp)
library(stringr)
library(ggplot2)
library(parallel)
library(doParallel)
library(randomForest)
library(caret)
library(concaveman)
library(sm)
library(pracma)
library(sf)
library(stats)
library(grDevices)
library(flowMagic)

#----- using template model with 1 template

# get path to directory with files to analyze
path_dir<-system.file("extdata/csv_files",package = "flowMagic")

# import data with labels that we use as template data.
list_data_ref<-import_reference_csv(path_results = path_dir,n_cores = 1)

# import data without labels
list_test_data<-import_test_set_csv(path_data = path_dir,n_cores = 1)

# Note that it is possible to provide also directly the paths to each file. See functions manual for additional
  details.

# data preprocessing to generate template model using first file as template
ref_train<-get_train_data(paths_file = list_data_ref[1],n_cores = 1) # we select first element of the imported list
  of dataframes

# generate the template model using out-of-the-bag validation
ref_model_info<-magicTrain(df_train = ref_train,n_cores = 1,train_model = "rf")

# perform automated gating (gates boundaries prediction step)
list_dfs_pred<-magicPred_all(list_test_data = list_test_data,magic_model = NULL,ref_data_train = ref_train,
  ref_model_info = ref_model_info,n_cores = 8)

# Note that providing the training set in the magicPred function is optional (ref_data_train = ref_train is
  optional).
# Providing the training set allows the user to calculate the target-template distance for each plot to analyze.

# list_dfs_pred contains a list of dataframes for each plot analyzed. In other words, it is a nested list (e.g.,
  downsampled dataset and original dataset with predicted labels for each plot). See the functions manual for the
  full list of dataframes returned.

# visualize gated data

df_temp<-list_dfs_pred[[1]]$df_test_original # dataframe of first gated plot

magicPlot(df = df_temp,type = "ML",size_points = 1)

magicPlot(df = df_temp,type = "dens",size_points = 1)

#----- using template model with multiple templates

# get path to directory with files to analyze
path_dir<-system.file("extdata/csv_files",package = "flowMagic")

# import data with labels that we use as template data.
list_data_ref<-import_reference_csv(path_results = path_dir,n_cores = 1)

# import data without labels
list_test_data<-import_test_set_csv(path_data = path_dir,n_cores = 1)

# Note that it is possible to provide also directly the paths to each file. See functions manual for additional
  details.

```

```

# data preprocessing for generate template model using multiple templates
ref_train<-get_train_data(paths_file = list_data_ref,n_cores = 1) # we select all elements of the imported list of
dataframes

# indices for leave-one-out cross-validation when training multiple templates.
list_inds_cross_val<-get_indices_cross_val(df_train = ref_train,n_cores = 8, train_inds = "leave_one_out",
val_inds = "leave_one_out")

# generate template model using leave-one-out cross validation validation.
ref_model_info<-magicTrain(df_train = ref_train,n_cores = 8,train_model = "rf",
list_index_train = list_inds_cross_val$inds_train,list_index_val = list_inds_cross_val$inds
_val)

# perform automated gating (gates boundaries prediction step)
list_dfs_pred<-magicPred_all(list_test_data = list_test_data,magic_model = NULL,ref_data_train = ref_train,
ref_model_info = ref_model_info,n_cores = 8)

# visualize gated data

df_temp<-list_dfs_pred[[1]]$df_test_original # dataframe of first gated plot

magicPlot(df = df_temp,type = "ML",size_points = 1)

magicPlot(df = df_temp,type = "dens",size_points = 1)

```

Example of correct script using the generalized model.

```

# load libraries

library(sp)
library(stringr)
library(ggplot2)
library(parallel)
library(doParallel)
library(randomForest)
library(caret)
library(concaveman)
library(sm)
library(pracma)
library(sf)
library(stats)
library(grDevices)
library(flowMagic)
library(flowMagic)

# The first step is to download the trained generalized model (Model A and list of Model B) from the Federated
Research Data Repository (FRDR) at the link:https://doi.org/10.20383/103.01352

# extract the tar.gz file and load the models.
model_a<-readRDS("./training_rf_index_3000train10val_2ntree_500points_100folds_31000_consensus_plots_pred_n_gates.
RData")
model_b_list<-readRDS("./list_models_all_n_gates.RData")

# get path to directory with files to analyze
path_dir<-system.file("extdata/csv_files",package = "flowMagic")

# import data without labels
list_test_data<-import_test_set_csv(path_data = path_dir,n_cores = 1)

# perform automated gating (gates boundaries prediction step)
list_dfs_pred<-magicPred_all(list_test_data = list_test_data,magic_model = model_b_list,
magic_model_n_gates = model_a,n_cores = 1)

# visualize gated data

df_temp<-list_dfs_pred[[1]]$df_test_original # dataframe of first gated plot

magicPlot(df = df_temp,type = "ML",size_points = 1)

magicPlot(df = df_temp,type = "dens",size_points = 1)

```

As mentioned previously, it is possible to generate your own generalized model by following the instructions provided in the appropriate section (see generalized model training section).

5 flowMagic workflow analyzing FCS files

This section demonstrates a complete workflow using the `flowMagic` package, starting from raw `.fcs` files, performing compensation and transformation, interactive manual gating using `magicGating`, generating template models, and performing automated gating across samples.

5.1 Manual Gating

First we demonstrate how to import raw FCS files, apply compensation and transformation, visualize the gated population, and perform interactive manual gating using the `magicGating` function included in the `flowMagic` package.

```
# Remember to load the libraries to read, visualize and analyse flow cytometry data, including flowMagic.
library(flowMagic)
library(flowCore)
library(ggplot2)
library(flowWorkspace)
library(devtools)
library(CytoML)

# Import FCS files from flowMagic extdata directory
path_dir <- system.file("extdata/fcs_files", package = "flowMagic")

fs <- read.flowSet(
  path = path_dir,
  transformation = FALSE,
  pattern = ".fcs"
)

# Create a GatingSet
gs <- GatingSet(fs)

# View sample names
sampleNames(gs)

# Load compensation matrix from extdata
path_comp_file <- system.file(
  "extdata/comp_matrix.csv",
  package = "flowMagic"
)

comp_matrix <- read.csv(path_comp_file, check.names = FALSE)

# Apply compensation to the GatingSet
gs <- compensate(gs, comp_matrix)

# Channels to be transformed using a logicle transform
channels_to_transform <- c(
  "FL1.A", "FL2.A", "FL3.A", "Viability_dye"
)

# Estimate transform based on the first sample
trans_list <- estimateLogicle(
  gs[[1]],
  channels = channels_to_transform
)

# Apply transformation
gs <- transform(gs, trans_list)

# visualize pre-processed data
flowMagic::magicPlot_fs(
  fs = fs,
  sample_id = 1,
  channel_x = "FSC.A",
  channel_y = "FSC.H"
)

# manual gate singlets ####

list_out <- magicGating(fs = fs, sample_id = c("sim_standard_gating_01.fcs"),
```

```

channel_x = "FSC.A",channel_y = "FSC.H",size_points=0.5)

df_1<-list_out$list_gated_data$sim_standard_gating_01.fcs

magicPlot(df = df_1) # visually check manually gated plot

# Apply the same polygonGate to all samples in the GatingSet
gs_pop_add(gs, list_out$list_poly_gates$sim_standard_gating_01.fcs, parent = "root", name = "Singlets")

# Recompute the gating
recompute(gs)

# plot gating tree
plot(gs)

# we can save the new GatingSet with the new gating information
# Note that the path to an empty directory is required.
# In this example, the exported_gs_directory is a new empty directory.
save_gs(gs = gs,path = "/home/rstudio/main/Data/sim_data/exported_gs")

```

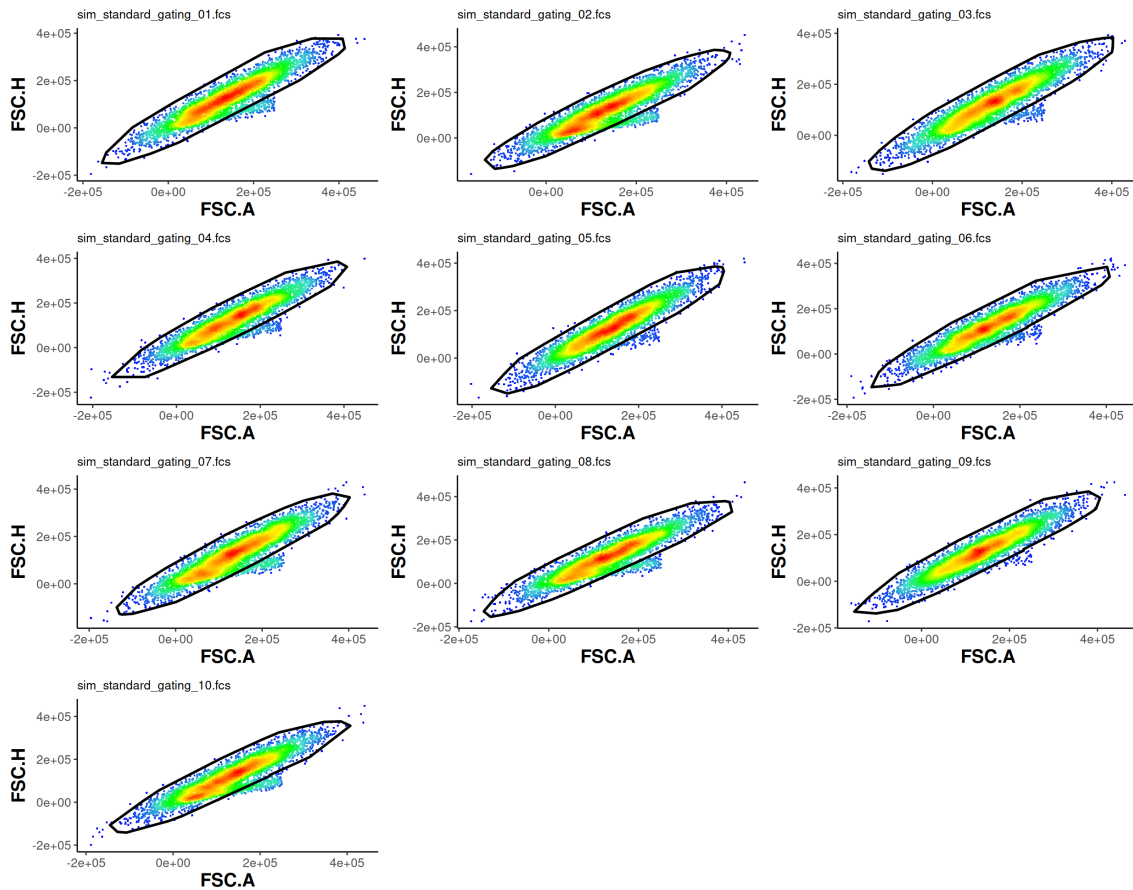


Figure 1: Manual Gating of Singlets (10 samples) using flowMagic gating framework. The plot was generated using the magic plot wrap function.

5.2 Automated Gating with template model

After manually defining a singlet gate, we proceed to identify live cells. This section shows: loading a previously saved `GatingSet`, manual template creation for live-cell gating, training a machine-learning template model, performing automated gating across all samples, and visualizing the results.

```

# Automated gating live cell using Template model ####

# import gs (can also be imported from flowJo or FCS express)
gs<-load_gs(path = "/home/rstudio/main/Data/sim_data/exported_gs")

sampleNames(gs)

```

```

ff<-flowMagic::get_flowframe_from_gs(gs = gs,node_name = "Singlets",sample_id = 1)

flowMagic::magicPlot_fs(fs = ff,channel_x = "Viability_dye",channel_y = "FSC.A")

# make templates

list_out_1<-magicGating(fs = gs,sample_id = c("sim_standard_gating_01.fcs"),
  channel_x = "Viability_dye",channel_y = "FSC.A",gs_node = "Singlets",label_pol = "1")

list_out_2<-magicGating(fs = gs,sample_id = c("sim_standard_gating_01.fcs"),
  channel_x = "Viability_dye",channel_y = "FSC.A",gs_node = "Singlets",label_pol = "2")

list_out_final<-merge_magicGating_labels(list_out_1 = list_out_1,list_out_2 = list_out_2)

# we convert these gated samples to a training set
ref_train<-get_train_data(paths_file = list_out_final)

# training step based on template data to generate template model

ref_model_info<-magicTrain(df_train = ref_train,train_model = "rf")

# get test data
list_test_data<-flowMagic::export_raw_gs_plots(gs = gs,node_name = "Singlets",channel_x = "Viability_dye",channel_y
  = "FSC.A",
  return_data = T)

# perform automated gating based on template model
list_dfs_pred<-magicPred_all(list_test_data = list_test_data,magic_model = NULL,ref_data_train = ref_train,
  ref_model_info = ref_model_info,n_cores = 1)

# export gated plots
exports_plots(list_gated_data = list_dfs_pred,path_output = "~/main/Data/results_sim_data")

# visualize all gated plots wrapped together
flowMagic::magic_plot_wrap(list_gated_data = list_dfs_pred,n_col_wrap = 3)

flowMagic::magic_plot_wrap(list_gated_data = list_dfs_pred,n_col_wrap = 3,
  size_points=0.5,size_title_x=15,size_title_y=15,size_axis_text=10)

# add gated live cells into Gating hierarchy
list_poly_gates<-flowMagic::flowmagic_pred_to_poly_gates(list_df = list_dfs_pred,gate_label = "Live_cells",pred_
  label = "1")

gs<-flowMagic::flowmagic_pred_to_gs(list_poly_gates = list_poly_gates,gs = gs,parent_node = "Singlets")

list_poly_gates<-flowmagic_pred_to_poly_gates(list_df = list_dfs_pred,gate_label = "Dead_cells",pred_label = "2")

gs<-flowMagic::flowmagic_pred_to_gs(list_poly_gates = list_poly_gates,gs = gs,parent_node = "Singlets")

save_gs(gs = gs,path = "/home/rstudio/main/Data/sim_data/exported_gs_2")

```

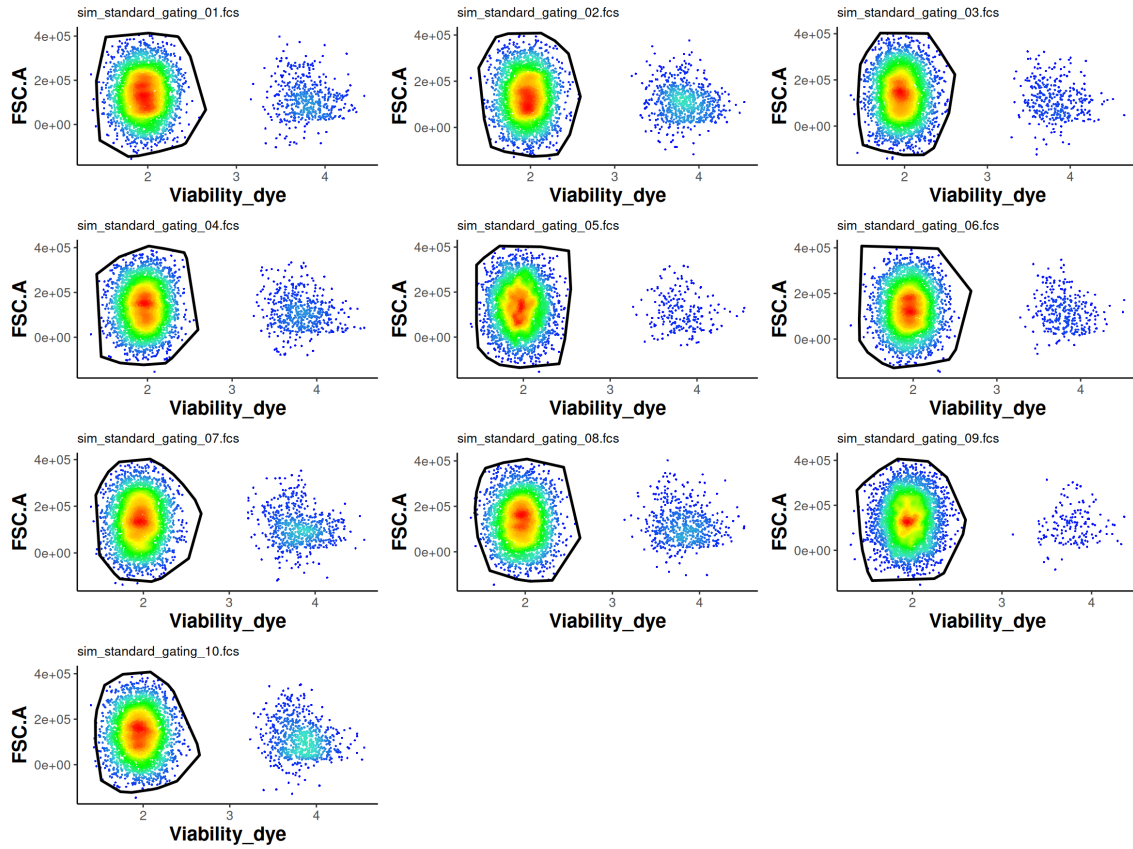


Figure 2: Automated Gating of Live cells (10 samples) using flowMagic template model. The plot was generated using the magic plot wrap function.

5.3 Automated Gating with generalized model

In this section, the same live cells previously gated with the template model will be gated using the generalized model.

```
# Automated gating live cells using Generalized model ####

# we need to reload the previous GatingSet we did not modify with the new Live cells gate
gs<-load_gs(path = "/home/rstudio/main/Data/sim_data/exported_gs")

# First, we need to load the generalized model (divided in model A and B) from the Federated Research Data
# Repository (FRDR).
# Once downloaded we need to load them in R using the readRDS() function.
model_a<-readRDS("/home/rstudio/main/GP_model/models_trained_n_gates_final/models_trained_to_predict_n_gates_final/
training_rf_index_3000train10val_2ntree_500points_100folds_31000_consensus_plots_pred_n_gates.RData")
model_b<-readRDS("~/main/GP_model/models_trained_n_gates_final/models_trained_to_predict_classes_final/list_models_
all_n_gates.RData")

# get test data
list_test_data<-flowMagic::export_raw_gs_plots(gs = gs,node_name = "Singlets",channel_x = "Viability_dye",channel_y
= "FSC.A",
                                             return_data = T)

# perform automated gating based on generalized model

# with a single predefined number of gates for all samples
list_dfs_pred<-magicPred_all(list_test_data = list_test_data,magic_model = model_b,magic_model_n_gates = 2,
                             n_cores = 8,n_points_per_plot = 15000)

# with a single predefined number of gates for selected samples
list_dfs_pred<-magicPred_all(list_test_data = list_test_data,sample_id = c(1,5,10),magic_model = model_b,magic_
model_n_gates = 2,
                             n_cores = 8,n_points_per_plot = 15000) # instead of positional indices, samples names
                             can also be indicated
```

```

# with different predefined number of gates based on the samples
# generating dataframe that pairs sample names with predefined number of gates.
n_gates_samples<-rep(2,length(list_test_data))
sample_name<-names(list_test_data)
n_gates_df<-as.data.frame(cbind(sample_name,n_gates_samples))
n_gates_df$n_gates_samples[5]<-1
n_gates_df$n_gates_samples[6]<-1
n_gates_df$n_gates_samples[9]<-1

list_dfs_pred<-magicPred_all(list_test_data = list_test_data,magic_model = model_b,magic_model_n_gates = 2,
                             n_gates_df=n_gates_df,
                             n_cores = 8,n_points_per_plot = 15000)

# with number of gates predicted by model A for samples

list_dfs_pred<-magicPred_all(list_test_data = list_test_data,magic_model = model_b,magic_model_n_gates = model_a,
                             n_cores = 8,n_points_per_plot = 15000,thr_dist = 0.2)

# visualize all gated plots wrapped together
flowMagic::magic_plot_wrap(list_gated_data = list_dfs_pred,n_col_wrap = 3,
                            size_points=0.5,size_title_x=15,size_title_y=15,size_axis_text=10)

# visualize density of events next to each axis.
magicPlot(list_dfs_pred$sim_standard_gating_05.fcs$df_test_original,show_marginals = T)

magicPlot(list_dfs_pred$sim_standard_gating_02.fcs$df_test_original,show_marginals = T)

# export gated plots
exports_plots(list_gated_data = list_dfs_pred,path_output = "~/main/Data/results_sim_data")

```

5.4 Automated Gating of whole pre-defined hierarchy

In this mode of execution, flowMagic automatically trains on each gating step of a predefined hierarchy and automatically gate the FCS files of interest based on the combinations of markers indicated in the gating hierarchy. The gating hierarchy information is contained within the GatingSet R object generated by the user using the flowWorkspace package. For example, the users can use flowDensity to gate a specific number of FCS files and then they store the gated results in a GatingSet object. Alternatively, the users can manually gate the FCS files using the FlowJo software and then they can export a WSP file that flowMagic will automatically convert into a GatingSet object. This GatingSet object can be used to automatically generate the template model for each gating step of the hierarchy. **Note: this mode of execution is compatible only with the template model. The generalized model cannot be used in this mode.**

First, the GatingSet needs to be imported.

```

gs_sample_gated<-import_gating_info(path="path/to/data")

gh_sample_gated_1<-gs_sample_gated[[1]] # first template

gh_sample_gated_2<-gs_sample_gated[[2]] # second template.

```

We also need to import the CSV files to analyze. Note that the FCS files need to have the same channel names. By default the first FCS is chosen as reference to check that all channel names are the same. FCS with channel names different from the chosen reference are excluded.

```

fs<-import_test_set_fcs(path= "/home/rstudio/final_data_test/HIPC_test_data/Myeloid_panel/test_fcs",n_samples = 1,
                        ref_f_n = 1)

```

Next, we get the list of training sets for each gating step of the hierarchy.

```

# import gating hierarchy from the GatingHierarchy object which contains the gating hierarchy information of the
# sample.
out<-get_hierarchy_all_pops(gh=gh_sample_gated_1,export_visnet = F)

# Extract all training data.
list_all_train_sets_1<-get_local_train_sets(gh=gh_sample_gated_1,hierarchical_tree=out$hierarchical_tree,
info_hierarchy=out)

list_all_train_sets_2<-get_local_train_sets(gh=gh_sample_gated_2,hierarchical_tree=out$hierarchical_tree,
info_hierarchy=out)

```

We also prepare the data to analyze.

```
list_all_test_sets<-get_test_sets(fs,gh=gh_sample_gated_1)
```

Finally, the user can execute the training of each training set and the prediction based on the hierarchy structure.

```
out_train<-magicTrain_hierarchy(list_train_sets = list_all_train_sets_1,n_cores = 8)
list_models<-out_train$list_models_sets_all_levels

list_gated_data<-magicPred_hierarchy(list_test_sets=list_all_test_sets,list_models_local = list_models,df_tree =
  out$df_tree,n_cores = 1)
```

The variable list_gated_data contains the labeled dataset for each gating step.

```
# extract gated results of the Singlets populations for first example
df_gated_1<-list_gated_data[[1]]$'level:6'$Singlets$gated_data

# Visualize gated population.
magicPlot(df=df_gated_1,type = "ML")
```

6 Advance functionalities

6.1 Generalized model training (only expert users)

This section is related to users with experience in training machine learning algorithms. Users with no machine learning experience can skip this section.

In order to generate the generalized model it is required to import the dataset to use for training. The get_train_data function generates the correctly formatted training set from the raw reference data, extracting the density features needed for the training. The input can either be the list of labeled dataframes (with the third column indicating the label assigned to each event) or the paths that lead to the labeled data. Since the data required to train the generalized model is usually very large, providing the paths may be the best option instead of importing the raw data into memory. The paths format can either be a vector of paths pointing directly to the labeled dataframe or a two-columns dataframe with the first column indicating the path to the expression data and the second column indicating the path to the labels of each event. Note that the data is normalized by default. Since the data for the generalized model is usually very large in size, it is suggested to perform a 500 points downsampling or similar to avoid overcoming the machine memory and speed the training. This is also the default option.

```
df_train<-get_train_data(df_paths = df_paths,n_cores = 1,n_points_per_plot = 500) # using dataframes of paths and
  500 points downsampling.

df_train<-get_train_data(paths_file = vec_paths,n_cores = 1, n_points_per_plot = 500) # using vector of paths to
  labeled dataframes and 500 points downsampling.
```

As mentioned before, the generalized model is composed of two models: Model A and Model B. In order to generate Model A, the users need to provide the data containing plots with a different number of gates. Then they will need to extract the cross-validation indices selecting the number of random plots in the training set for each number of gates and the number of random plots in the validation set for each number of gates. The n_folds argument indicates the number of times this process is repeated. Finally, the users will need to execute the training indicating the number of gates as response variable.

```
# generate cross-validation indices using 1000 training plots and 50 validation plots for 50 repetitions.
list_inds_cross_val<-get_indices_cross_val(df_train = df_train,n_cores = 4,train_inds = "rand_set_n_gates_info",n_
  train_plots = 1000,n_folds = 50,val_inds = "rand_set_n_gates_info",n_val_plots = 50)

# generate the generalized model using random forest.
random_forest_model<-magicTrain(df_train = df_train,n_cores = 4,train_model = "rf",list_index_train = list_inds_
  cross_val$inds_train,
list_index_val = list_inds_cross_val$inds_val,n_tree = 10,type_y = "n_gates_info")
```

In order to generate Model B for the specific number of gates, the users need to provide the data related to plots having only a specific number of gates.

```
# Selecting only plots with 2 gates.
df_train_ngates_selected<-df_train[df_train$n_gates_info==2,]
row.names(df_train_ngates_selected)<-NULL
```

Then, the cross-validation indices need to be extracted indicating the number of random plots in the training set and the number of plots in the validation set. Finally, the users will need to execute the training indicating the gates assignment (the classes column) as response variable.

```
# get cross-validation indices using 300 training plots and 5 validation plots for 50 repetitions.
list_inds_cross_val<-get_indices_cross_val(df_train = df_train_ngates_selected,n_cores = 8, train_inds = "rand_set_
num",n_train_plots = 300,n_folds = 100, seed = 50,val_inds = "rand_set_num",n_val_plots = 5)

# generate the generalized model for the gates boundaries of the specific number of classes.

random_forest_model<-magicTrain(df_train = df_train_ngates_selected, n_cores = 4, train_model = "rf", list_index_
train = list_inds_cross_val$inds_train, list_index_val = list_inds_cross_val$inds_val,n_tree = 10,type_y = "
classes")
```

The users can also train a model considering the gates boundaries for all numbers of gates. In this case, they need to provide the original complete dataframe containing the plots with all numbers of gates. The users will need to extract the cross-validation indices selecting the number of random plots in the training set for each number of gates and the number of random plots in the validation set for each number of gates.

```
# generate cross-validation indices using 1000 training plots and 50 validation plots for 50 repetitions.
list_inds_cross_val<-get_indices_cross_val(df_train = df_train,n_cores = 4,train_inds = "rand_set_n_gates_info",n_
train_plots = 1000,n_folds = 50,val_inds = "rand_set_n_gates_info",n_val_plots = 50)

# generate the generalized model for the gates boundaries for all number of gates.
random_forest_model<-magicTrain(df_train = df_train, n_cores = 4, train_model = "rf", list_index_train = list_inds_
cross_val$inds_train, list_index_val = list_inds_cross_val$inds_val,n_tree = 10,type_y = "classes")
```