flowSim: improving the training sets and machine learning models in flow cytometry data analysis

Sebastiano Montante

September 15, 2022

Abstract

This pdf describes the usage of the main flowSim functions and reports examples of typical flowSim scripts

1 Introduction

flowSim is a tool designed to visualize, detect and remove highly redundant information in large FCM training sets, increasing the performance of ML algorithms by reducing overfitting while at the same time decreasing computational time for training. The tool performs near duplicate image detection (NDD) by combining community detection algorithms with the density analysis of the marker expression values

2 Installation

The easiest way to install flowSim is directly from github:

```
install_github("semontante/flowSim")
```

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

```
install("path/to/local_dir")
```

3 Input

The input of flowSim is a directory containing the bivariate marker expression of the images under analysis. In particular, the marker expression of each image must be reported in a csv file whose first and second column report the expression of, respectively, the first and second marker. 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

4 Identification and removal of similar images

First, we need to calculate the pairwise distance matrix of all input files under analysis:

```
# get the directory path there the csv files are stored

fpath <- system.file("extdata", package="flowSim")

# get similarity scores matrix for all files (small datasets, < 500 files)
out<-get_similarity_all_plots(path_dir = fpath)
```

Based on this distance matrix we generate a network of nodes, where each node is a file under analysis, and we apply a community detection algorithm to identify the clusters of similar files:

```
# Execute igraph clustering

list_output<-gen_igraph_network(similarity_output = out,method_clust = "louvain") # louvain clustering
igraph_network<-list_output$igraph_network
partition_vec<-list_output$partition
# generate visnetwork
set.seed(1234) # set seed to get same groups colors
visnetdata<-gen_visnetwork_data(igraph_network)
# heterogeneity score (Optional)
heterogeneity_score<-get_heterogeneity_score(igraph_network,partition_vec=partition_vec,visnetdata = visnetdata)
```

The user can optionally visualize the network:

```
# plot full network
plot_visnet(visnetdata)
```

We can now filter out the similar files and export the final results. The user has several options here. (1) It is possible to get only a dataframe reporting the names of the files, (2) to export all the files to keep in a directory or (3) to export all the files before the filtering process:

```
# (2) export the plots of all files based on groups
exports_plots(visnetdata = visnetdata,path_expr_data = fpath, path_output = "path/to/output_dir")

# (3) export the plots of only the files to keep
info_exporting<-exports_filtered_plots(visnetdata = visnetdata,df_features = out$df_features, path_expr_data = fpath,

# (1) get only a dataframe reporting the files to keep from each group.

info_exporting<-exports_filtered_plots(visnetdata = visnetdata,df_features = out$df_features, path_expr_data = fpath,
```

In the (2) and (3) the output directory will contain a folder for each group identified.

5 Visualization options

There are 3 different type of network visualization. The user can visualize all the input files organized in custers and all the isolated files (full network visualization):

```
# plot full network
plot_visnet(visnetdata)
# plot full network setting the size of nodes
plot_visnet(visnetdata,size_nodes=25)
```

The user can also select only a partial area of the network to visualize and the isolated files can be removed ("partial network visualization"). This is particularly useful for large networks.

```
# plot network without the isolated files
plot_visnet(visnetdata,remove_single_files = T)
# plot only specific groups
plot_visnet(visnetdata,select_group = c("1"))
plot_visnet(visnetdata,select_group = c("1","3","7","10"))
plot_visnet(visnetdata,select_group = c("3","7","10"))
```

Finally, for very large networks (¿ 4000 files), it is not possible to plot the full network at once. It is useful in this case to plot a contracted version of the network, where each node is a group and each edge is proportional to the number of connections between the groups.

```
# generate contracted data
visnetdata_contracted<-gen_visnetwork_data(igraph_network,contract_net = T)
# plot contracted network
plot_visnet(visnetdata_contracted)
```

6 small vs large datasets, parameters tuning

The V1 algorithm is designed to work for small datasets (¡ 500 files) and it is also the most accurate version.

```
# get similarity scores matrix for all files (small datasets, < 500 files)
out<-get_similarity_all_plots(path_dir = fpath)
```

There are several parameters the user can set to personalize the similarity output. The tolerance parameter control the tolerance of differences within each group. The defaul value for the V1 mode is 0.6. The user can change this value to regulate the level of differences within each group:

```
# Set tolerance to 0.7. Default is 0.6
out<-get_similarity_all_plots(path_dir = fpath, thr_score = 0.7)
```

It is also possible to set the number of cores to use

```
# Use 4 cores.
out<-get_similarity_all_plots(path_dir = fpath, n_cores=4)
```

The V2 algorithm is designed to work with large datasets. It is faster, it consumes less memory, but it is also less accurate. It has the same parameters of V1 and an additional parameter to control the level of fragmentation of the dataset. As the fragmentation parameter increases, the time of execution and memory load decreases, but the accuracy also decreases. Up to 2000 files the fragmentation parameter is automatically estimated, above 2000 files it needs to be manually set. The default tolerance parameter for the V2 algorithm is 0.9.

```
# dataset broken in 5 fragments
out<-get_similarity_all_plots_v2(path_dir = fpath, n_batches = 5)
# dataset broken in 100 fragments
out<-get_similarity_all_plots_v2(path_dir = fpath, n_batches = 100)
```

It is possible to increase the accuracy of the V2 algorithm, by performing several cycles of execution. Each cycle is performed on the files selected from the previous cycle. In this way, the residual homogeneity generated from the previous cycles is progressively reduced.

```
# We execute flowSim as usual.
out<-get_similarity_all_plots_v2(path_dir = fpath)

list_output<-gen_igraph_network(similarity_output = out,method_clust = "louvain")
igraph_network<-list_output$igraph_network
partition_vec<-list_output$partition
set.seed(1234)
visnetdata<-gen_visnetwork_data(igraph_network)

info_exporting<-exports_filtered_plots(visnetdata = visnetdata,df_features = out$df_features, path_expr_data = fpath,

# The filtered files are the files of the next cycle
files_new_cycle<-unlist(strsplit(info_exporting$files_filtered,";"))

inds_selected_files<-get_inds_files_selected(path_dir = fpath, files_selected = files_new_cycle)


# We execute flowSim on the filtered files
out<-get_similarity_all_plots_v2(n_samples=inds_selected_files,path_dir = fpath)
```

Both V1 and V2 have the same exporting step, which can be regulated by the filtering parameter. The filtering parameter controls how many files are kept in the final filtered dataset. The default is 2.1. Decreasing this parameter, decreases the number of files removed, so it increases the number of files (i.e., patterns) kept in the final filtered dataaset.

```
# default threshold: 2.1
info_exporting<-exports_filtered_plots(visnetdata = visnetdata,df_features = out$df_features, path_expr_data = fpath,

# Filtering threshold set to 1.2. More patterns preserved.
info_exporting<-exports_filtered_plots(visnetdata = visnetdata,df_features = out$df_features, path_expr_data = fpath,
```