# Matlab for cellcognize data:

function features = CellcognizeCommunity(patha,pathb,comm,features,output1)

%% CellCognize%

% Script de base pour creer un model predictif et les données associées

%

%

%% import file a and b

if exist('mynn.m', 'file')==2

delete('mynn.m');

end

fprintf(patha)

fprintf(pathb)

if strcmp(patha(end-3:end),'.fcs')

a= fca\_readfcs(patha);

b=fca\_readfcs(pathb);

else

a=readtable(patha);

a=table2array(a);

b=readtable(pathb);

b=table2array(b);

end

standards ={a,b};

filtered\_standards={a,b}; % Data without gating and log = for cellcognize data already treated

% Select the column according to the list:

%%

%features=["FSC-A","FSC-W","FSC-H","SSC-A","SSC-W","SSC-H","FL1-A","FL1-W","FL1-H"]

%%

features

features= string(features);

%features=cell2mat(features)

feat=zeros(1,7);

%feat=[find(features=='FSC-H'),find(features=='SSC-H'),find(features=='FITC-H'),find(features=='FSC-A'),find(features=='SSC-A'),find(features=='FITC-H'),find(features=='Width')];

feat(1,1)=find(features=='FSC-H');

feat(1,2)=find(features=='SSC-H');

feat(1,4)=find(features=='FSC-A');

feat(1,5)=find(features=='SSC-A');

if ~isempty(find(features=='FITC-H'))

feat(3)=find(features=='FITC-H');

feat(6)=find(features=='FITC-A');

elseif isempty(find(features=='FL1-H'))

feat(3)=0;

feat(6)=find(features=='FITC-A');

else

feat(3)=find(features=='FL1-H');

feat(6)=find(features=='FL1-A');

end

if ~isempty(find(features=='Width'))

feat(7)=find(features=='Width');

else

feat(7)=0

end

%%

for i =1:2

final\_files1=standards{i};

scales1= final\_files1(:,feat(1)); %FSC-H

scales2= final\_files1(:,feat(2)); %SSC-H

if feat(3)==0

scales3=final\_files1(:,feat(6));

scales3(scales3>=0)=200;

else

scales3= final\_files1(:,feat(3));

end

%scales7= final\_files1(:,feat(7)); %Width

%scales3= final\_files1(:,feat(3)); %FITC-H

scales4= final\_files1(:,feat(4)); %FSC-A

scales5= final\_files1(:,feat(5)); %SSC-A

scales6= final\_files1(:,feat(6)); %FITC-A

if feat(7)==0

scales7=final\_files1(:,feat(6));

scales7(scales7>=5)=20;

else

scales7= final\_files1(:,feat(7)); %Width

end

scales=horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

standards{i}=scales;

end

%% Import file into a matrix such as:

%load('final\_file\_merged\_2019.mat');

% only taking the folowing colums, since they selected only these ones

%% Filter the files to within the lower and upper boundary thresholds, for each of the seven FCM channels. Order of the FCM channels in the files is:

% Column 1= FSC-H

% Column 2= SSC-H

% Column 3= FITC-H

% Column 4= FSC-A

% Column 5= SSC-A

% Column 6= FITC-A

% Column 7= Width

%Define the min and maxdata values for the filtering. Not sure we should

%filter the data since I didn't

mindata1=100;

maxdata1=4000000;

mindata2=100;

maxdata2=4000000;

mindata3=100;

maxdata3=500000;

mindata4=100;

maxdata4=2000000;

mindata5=100;

maxdata5=2000000;

mindata6=100;

maxdata6=1000000;

mindata7=10;

maxdata7=2000;

for i=1:2

final\_files1=standards{i};

%final\_files1\_filtered=final\_files1;

final\_files1\_filtered=final\_files1(final\_files1(:,1)<maxdata1,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,1)>mindata1,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,2)<maxdata2,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,2)>mindata2,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,3)<maxdata3,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,3)>mindata3,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,4)<maxdata4,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,4)>mindata4,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,5)<maxdata5,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,5)>mindata5,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,6)<maxdata6,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,6)>mindata6,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,7)<maxdata7,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,7)>mindata7,:);

%Do the log10-transformation for each column specifically.

scales1=log10(final\_files1\_filtered(:,1));

scales2=log10(final\_files1\_filtered(:,2));

scales3=log10(final\_files1\_filtered(:,3));

scales4=log10(final\_files1\_filtered(:,4));

scales5=log10(final\_files1\_filtered(:,5));

scales6=log10(final\_files1\_filtered(:,6));

scales7=log10(final\_files1\_filtered(:,7));

%Regroup the columns back into one file.

scales=horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

%filtered\_standards{i}=scales;

end;

%% Data annotation:

%Gate the different standards to either individual or multiple subpopulations, depending on the 2D plot aspect of FCS-H vs SSC-H vs FITC-H (example in Fig. S1). Bead standards do not require further gating.

%Rename the bead standards

%species1=standard{1};

%species2=standard{2};

%Now combine the filtered gated standard files into one. This is the 32-standard input file for the ANN model.

%filtered\_standards={species1,species2};

%% Section 2. Artificial neural network reconstruction.

% The filtered and gated data sets of the standards (n ~ 3 × 105 to 1.5 × 106 events per standard) were used as input for the development of ANN models. The datasets were randomly subsampled to 10,000 events per standard using the datasample function (Matlab v. R2017a). Crucially, the lower and upper boundary thresholds imposed during the filtering process for each of the seven FCM parameters were added as two data points (“anchors”) per parameter to the first subsampled standard. This process of ‘anchoring’ was essential for the subsequent machine-learning algorithm. Subsampled anchored datasets were concatenated and used as input into the ANN model, during which they were further scaled (between ?1 and 1, hence the necessity to add the anchors) and randomly divided using Dividerand (Matlab v. R2017a ) into three blocks: a training set (50% of the data), a validation set (25%) and a testing set (25%).

% The ANN architecture consisted of a feed-forward back-propagation algorithm with one input, one hidden and one output layer. The input layer contained 7 nodes (corresponding to the 7 FCM parameters), whereas the output layer contained 5 (for the preliminary three-strain experiment) or 32 nodes (one for each of the standards in the full set). Input nodes were connected to the hidden layer by the sigmoid function (Matlab v. 2017a), whereas the hidden layer nodes (20) were connected to the output by the softmax transfer function (Matlab v. 2017a). The input matrix was trained using the trainscg function (Matlab v. 2017a) in a 1000-cycle of training, validation and testing (performance goal = 0 | time = Inf | min grad = 10-6 | max fail = 6). Performance of the ANN was evaluated by crossentropy. The outcome of the ANN model is a learned linear equation, termed the ANN classifier, describing the correlations between input parameters and the five (proof-of-concept experiment, ANN-5) or 32 classes of the standard dataset (ANN-32). The process of subsampling, anchoring, concatenation and training was repeated five times independently on the full datasets, generating five slightly different ANN classifiers. The performance of the ANN classifiers was assessed on the basis of confusion matrices (Matlab v. 2017a), representing predicted versus actual events for the complete in silico mixed set of standards, and the false prediction rate (as shown in Fig. S2).

%2.1. Subsampling and anchoring

%Data were first randomly subsampled to same number of events. File array name from previous scaling was 'filtered\_standards'.

%path /Files\_for\_Zenodo/FCM\_files

standard\_normz=[];

sample\_size=5000;% training siza, 20000 will not be enough so we selected 5000

for i=1:length(filtered\_standards)

standard\_normz{i,1}=datasample(filtered\_standards{1,i},sample\_size,1);

end;

standard\_normz=standard\_normz';

%%

%alternatively: subsample to n = 5000. This file is saved as 'standard\_normz\_restricted\_32.mat' for section 3.7

%Add the line with the anchors for proper and consistent scaling throughout all data sets.

% Do not no how to change that if I need to. 7x7

anchors=[2,2,2,2,2,2,1;6.6,6.6,5.7,6.3,6.3,6.0,3.3];

%anchors=[2,2,2,2,2,2;6.6,6.6,5.7,6.3,6.3,6.0];

standard\_normz{1}=vertcat(anchors,standard\_normz{1});

%%

%2.2. ANN selection, training and validation

%Continue ANN with subsampled data set (standard\_normz); here 5000 events per standard.

file\_length\_final=cellfun(@length, standard\_normz);

%%

file\_size=file\_length\_final;

file\_length\_final=[1 file\_length\_final];

file\_length\_final=cumsum(file\_length\_final);

input=vertcat(standard\_normz{:});

output=zeros(length(input),length(standard\_normz));

for i=1:length(file\_length\_final)-1

output(file\_length\_final(i):file\_length\_final(i+1)-1,i)=1;

end

input=input';

output=output';

x = input;

t = output;

%%

%Choose a Training Function

trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.

%Create a Pattern Recognition Network

hiddenLayerSize = 20;

net = patternnet(hiddenLayerSize);

%Setup Division of Data for Training, Validation, Testing

net.divideFcn = 'dividerand'; % Divide data randomly

net.divideMode = 'sample'; % Divide up every sample

net.divideParam.trainRatio = 50/100;

net.divideParam.valRatio = 25/100;

net.divideParam.testRatio = 25/100;

%Choose a Performance Function

%For a list of all performance functions type: help nnperformance

net.performFcn = 'crossentropy'; % Cross-Entropy

%Choose Plot Functions

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...

'plotconfusion', 'plotroc'};

%Train the Network

[net,tr] = train(net,x,t);

%Test the Network

y = net(x);

e = gsubtract(t,y);

performance = perform(net,t,y);

tind = vec2ind(t);

yind = vec2ind(y);

percentErrors = sum(tind ~= yind)/numel(tind);

%Recalculate Training, Validation and Test Performance

trainTargets = t .\* tr.trainMask{1};

valTargets = t .\* tr.valMask{1};

testTargets = t .\* tr.testMask{1};

trainPerformance = perform(net,trainTargets,y)

valPerformance = perform(net,valTargets,y)

testPerformance = perform(net,testTargets,y)

%figure, plotconfusion(t,y)

strcat(output1,'.fig')

saveas(plotconfusion(t,y),strcat(output1,'.fig'))

genFunction(net,'mynn.m');

%% Section 3. CellCognize testing of standard-mixed communities.

% In a first proof-of-concept experiment, we cultured E. coli MG1655, P. veronii and A. johnsonii individually to stationary phase, diluted cultures 1:1000 in PBS, and measured cells by FCM after staining with Sybr Green I either individually, or in different mixtures of all three strains combined. Individual and mixture data sets were analyzed with CellCognize using a set of five replicate ANN-5 classifiers, comparing expected added cell numbers of each of the three strains with their assigned class attributions from the ANN-5 classifiers.

% 3.1 Preparing a limited ANN with five standards only.

% data import. Read filtered, log-transformed and gated E. coli, P. veronii and A. johnsonii data files. This has five datasets

% path: /Files\_for\_Zenodo/FCM\_files/MIX\_experiment\_ACL\_AJH\_PVR/ filtered\_standards\_AJH\_ECL\_PVE.mat

%filtered\_standards={AJH1,ECL1,ECL2,PVR1,PVR2};

% Continue training, validating and testing ANN-5 with subsampled data set; here 5000 events per sample; as in sections 2.1 and 2.2

% Example output classifier file saved as /Files\_for\_Zenodo/NN\_file\_example/NNfunction\_wide\_anchor\_normz\_filtered\_trimix\_310819.m

%

% 3.2 Analyze the synthetic community mixtures.

%

%Read in and treat data files of either cultures alone, or in combinations. Same folder: /Files\_for\_Zenodo/FCM\_files/MIX\_experiment\_ACL\_AJH\_PVR

%as example: combination of 30/10/10 ECL/AJH/PVE

SP1=readtable(comm);% TODO change for the community file

%SP1=fca\_readfcs('t0\_A\_A-B\_SGPI\_repl1.fcs');

SP1=table2array(SP1);

size(SP1)

feat

scales1= SP1(:,feat(1)); %FSC-H

scales2= SP1(:,feat(2)); %SSC-H

if feat(3)==0

scales3=SP1(:,feat(6));

scales3(scales3>=0)=200;

else

scales3= SP1(:,feat(3));

end

%scales3= SP1(:,feat(3)); %FITC-H

scales4= SP1(:,feat(4)); %FSC-A

scales5= SP1(:,feat(5)); %SSC-A

scales6= SP1(:,feat(6)); %FITC-A

if feat(7)==0

scales7= SP1(:,feat(6)); %Width

scales7(scales7>=0)=20;

else

scales7= SP1(:,feat(7)); %Width

end

comFile= horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

input\_community=comFile;

size(comFile)

%community=comFile

community=comFile(comFile(:,1)<maxdata1,:);

community=community(community(:,1)>mindata1,:);

community=community(community(:,2)<maxdata2,:);

community=community(community(:,2)>mindata2,:);

community=community(community(:,3)<maxdata3,:);

community=community(community(:,3)>mindata3,:);

ommunity=community(community(:,4)<maxdata4,:);

community=community(community(:,4)>mindata4,:);

community=community(community(:,5)<maxdata5,:);

community=community(community(:,5)>mindata5,:);

community=community(community(:,6)<maxdata6,:);

community=community(community(:,6)>mindata6,:);

community=community(community(:,7)<maxdata7,:);

community=community(community(:,7)>mindata7,:);

size(community)

%community=comFile;

%has 8 columns, remove column 8 check that for each file

scales1=log10(community(:,1));

scales2=log10(community(:,2));

scales3=log10(community(:,3));

scales4=log10(community(:,4));

scales5=log10(community(:,5));

scales6=log10(community(:,6));

scales7=log10(community(:,7));

%input\_community=horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

innput\_community=SP1

size(input\_community)

%add anchor line

%anchors=[2,2,2,2,2,2;6.6,6.6,5.7,6.3,6.3,6.0];

anchors=[2,2,2,2,2,2,1;6.6,6.6,5.7,6.3,6.3,6.0,3.3];

input\_community=vertcat(anchors,input\_community);

size(input\_community)

%transpose numeric array to be conform to NN input

input\_community=input\_community';

size(input\_community);

% run NN functions without thresholding

% vec2ind means that the index of the row is find where the value of 1 occurs. In this table it is the value that is rounded up to 1!

% NN functions are in switchdrive/final\_files/Community\_diversity\_analysis/NN functions

%make a table with the classes (1-5), a '0' (for the non-classified) - and one extra (33) that works as an anchor to fill the list properly (…)

classes=(0:3);

empty\_class=[3;0];

%NN function (1)

output\_restricted\_anchor1= net(input\_community); %available in example my nn

output\_restricted\_anchor\_final1=vec2ind(output\_restricted\_anchor1)

membership\_restricted1 = histc(output\_restricted\_anchor\_final1,unique(output\_restricted\_anchor\_final1));

classes\_restricted1=unique(output\_restricted\_anchor\_final1)

final\_community1= [classes\_restricted1;membership\_restricted1];

%now add the empty class to the final community

finalComm1=[final\_community1,empty\_class]

%apply a logical function to find corresponding values in the list with all categories in the file 'classes'

[lic,loc]=ismember(classes,finalComm1(1,:));

results1(2,lic)=finalComm1(2,loc(lic));

results1(1,:)=classes;

%produce final summary community and save as .csv. Modify the path if necessary.

final\_results=vertcat(results1)

T=array2table(final\_results','VariableNames',{'Class1','Count1'})

%write results to table, as example:

writetable(T,strcat(output1,'.csv'));% TODO

end

# Matlab for other data:

function features = CellcognizeCommunity(patha,pathb,comm,features,output1)

%% CellCognize%

% Script de base pour creer un model predictif et les données associées

%

%

%% import file a and b

if exist('mynn.m', 'file')==2

delete('mynn.m');

end

if strcmp(patha(end-3:end),'.fcs')

a= fca\_readfcs(patha);

b=fca\_readfcs(pathb);

else

a=readtable(patha);

a=table2array(a);

b=readtable(pathb);

b=table2array(b);

end

standards ={a,b};

%filtered\_standards={a,b}; % Data without gating and log = for cellcognize data already treated

% Select the column according to the list:

%%

%features=["FSC-A","FSC-W","FSC-H","SSC-A","SSC-W","SSC-H","FL1-A","FL1-W","FL1-H"]

%%

features

features= string(features);

%features=cell2mat(features)

feat=zeros(1,7);

%feat=[find(features=='FSC-H'),find(features=='SSC-H'),find(features=='FITC-H'),find(features=='FSC-A'),find(features=='SSC-A'),find(features=='FITC-H'),find(features=='Width')];

feat(1,1)=find(features=='FSC-H');

feat(1,2)=find(features=='SSC-H');

feat(1,4)=find(features=='FSC-A');

feat(1,5)=find(features=='SSC-A');

if ~isempty(find(features=='FITC-H'))

feat(3)=find(features=='FITC-H');

feat(6)=find(features=='FITC-A');

elseif isempty(find(features=='FL1-H'))

feat(3)=0;

feat(6)=find(features=='FITC-A');

else

feat(3)=find(features=='FL1-H');

feat(6)=find(features=='FL1-A');

end

if ~isempty(find(features=='Width'))

feat(7)=find(features=='Width');

else

feat(7)=0;

end

%%

for i =1:2

final\_files1=standards{i};

scales1= final\_files1(:,feat(1)); %FSC-H

scales2= final\_files1(:,feat(2)); %SSC-H

if feat(3)==0

scales3=final\_files1(:,feat(6));

scales3(scales3>=0)=200;

else

scales3= final\_files1(:,feat(3));

end

%scales7= final\_files1(:,feat(7)); %Width

%scales3= final\_files1(:,feat(3)); %FITC-H

scales4= final\_files1(:,feat(4)); %FSC-A

scales5= final\_files1(:,feat(5)); %SSC-A

scales6= final\_files1(:,feat(6)); %FITC-A

if feat(7)==0

scales7=final\_files1(:,feat(6));

scales7(scales7>=5)=20;

else

scales7= final\_files1(:,feat(7)); %Width

end

scales=horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

standards{i}=scales;

end

%% Import file into a matrix such as:

%load('final\_file\_merged\_2019.mat');

% only taking the folowing colums, since they selected only these ones

%% Filter the files to within the lower and upper boundary thresholds, for each of the seven FCM channels. Order of the FCM channels in the files is:

% Column 1= FSC-H

% Column 2= SSC-H

% Column 3= FITC-H

% Column 4= FSC-A

% Column 5= SSC-A

% Column 6= FITC-A

% Column 7= Width

%Define the min and maxdata values for the filtering. Not sure we should

%filter the data since I didn't

mindata1=100;

maxdata1=4000000;

mindata2=100;

maxdata2=4000000;

mindata3=100;

maxdata3=500000;

mindata4=100;

maxdata4=2000000;

mindata5=100;

maxdata5=2000000;

mindata6=100;

maxdata6=1000000;

mindata7=10;

maxdata7=2000;

for i=1:2

final\_files1=standards{i};

%final\_files1\_filtered=final\_files1;

final\_files1\_filtered=final\_files1(final\_files1(:,1)<maxdata1,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,1)>=mindata1,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,2)<maxdata2,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,2)>=mindata2,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,3)<maxdata3,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,3)>=mindata3,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,4)<maxdata4,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,4)>=mindata4,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,5)<maxdata5,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,5)>=mindata5,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,6)<maxdata6,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,6)>=mindata6,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,7)<maxdata7,:);

final\_files1\_filtered=final\_files1\_filtered(final\_files1\_filtered(:,7)>=mindata7,:);

%Do the log10-transformation for each column specifically.

scales1=log10(final\_files1\_filtered(:,1));

scales2=log10(final\_files1\_filtered(:,2));

scales3=log10(final\_files1\_filtered(:,3));

scales4=log10(final\_files1\_filtered(:,4));

scales5=log10(final\_files1\_filtered(:,5));

scales6=log10(final\_files1\_filtered(:,6));

scales7=log10(final\_files1\_filtered(:,7));

%Regroup the columns back into one file.

scales=horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

filtered\_standards{i}=scales;

end;

%% Data annotation:

%Gate the different standards to either individual or multiple subpopulations, depending on the 2D plot aspect of FCS-H vs SSC-H vs FITC-H (example in Fig. S1). Bead standards do not require further gating.

%Rename the bead standards

%species1=standard{1};

%species2=standard{2};

%Now combine the filtered gated standard files into one. This is the 32-standard input file for the ANN model.

%filtered\_standards={species1,species2};

%% Section 2. Artificial neural network reconstruction.

% The filtered and gated data sets of the standards (n ~ 3 × 105 to 1.5 × 106 events per standard) were used as input for the development of ANN models. The datasets were randomly subsampled to 10,000 events per standard using the datasample function (Matlab v. R2017a). Crucially, the lower and upper boundary thresholds imposed during the filtering process for each of the seven FCM parameters were added as two data points (“anchors”) per parameter to the first subsampled standard. This process of ‘anchoring’ was essential for the subsequent machine-learning algorithm. Subsampled anchored datasets were concatenated and used as input into the ANN model, during which they were further scaled (between ?1 and 1, hence the necessity to add the anchors) and randomly divided using Dividerand (Matlab v. R2017a ) into three blocks: a training set (50% of the data), a validation set (25%) and a testing set (25%).

% The ANN architecture consisted of a feed-forward back-propagation algorithm with one input, one hidden and one output layer. The input layer contained 7 nodes (corresponding to the 7 FCM parameters), whereas the output layer contained 5 (for the preliminary three-strain experiment) or 32 nodes (one for each of the standards in the full set). Input nodes were connected to the hidden layer by the sigmoid function (Matlab v. 2017a), whereas the hidden layer nodes (20) were connected to the output by the softmax transfer function (Matlab v. 2017a). The input matrix was trained using the trainscg function (Matlab v. 2017a) in a 1000-cycle of training, validation and testing (performance goal = 0 | time = Inf | min grad = 10-6 | max fail = 6). Performance of the ANN was evaluated by crossentropy. The outcome of the ANN model is a learned linear equation, termed the ANN classifier, describing the correlations between input parameters and the five (proof-of-concept experiment, ANN-5) or 32 classes of the standard dataset (ANN-32). The process of subsampling, anchoring, concatenation and training was repeated five times independently on the full datasets, generating five slightly different ANN classifiers. The performance of the ANN classifiers was assessed on the basis of confusion matrices (Matlab v. 2017a), representing predicted versus actual events for the complete in silico mixed set of standards, and the false prediction rate (as shown in Fig. S2).

%2.1. Subsampling and anchoring

%Data were first randomly subsampled to same number of events. File array name from previous scaling was 'filtered\_standards'.

%path /Files\_for\_Zenodo/FCM\_files

standard\_normz=[];

sample\_size=5000;% training siza, 20000 will not be enough so we selected 5000

for i=1:length(filtered\_standards)

standard\_normz{i,1}=datasample(filtered\_standards{1,i},sample\_size,1);

end;

standard\_normz=standard\_normz';

%%

%alternatively: subsample to n = 5000. This file is saved as 'standard\_normz\_restricted\_32.mat' for section 3.7

%Add the line with the anchors for proper and consistent scaling throughout all data sets.

% Do not no how to change that if I need to. 7x7

anchors=[2,2,2,2,2,2,1;6.6,6.6,5.7,6.3,6.3,6.0,3.3];

%anchors=[2,2,2,2,2,2;6.6,6.6,5.7,6.3,6.3,6.0];

standard\_normz{1}=vertcat(anchors,standard\_normz{1});

%%

%2.2. ANN selection, training and validation

%Continue ANN with subsampled data set (standard\_normz); here 5000 events per standard.

file\_length\_final=cellfun(@length, standard\_normz);

%%

file\_size=file\_length\_final;

file\_length\_final=[1 file\_length\_final];

file\_length\_final=cumsum(file\_length\_final);

input=vertcat(standard\_normz{:});

output=zeros(length(input),length(standard\_normz));

for i=1:length(file\_length\_final)-1

output(file\_length\_final(i):file\_length\_final(i+1)-1,i)=1;

end

input=input';

output=output';

x = input;

t = output;

%%

%Choose a Training Function

trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.

%Create a Pattern Recognition Network

hiddenLayerSize = 20;

net = patternnet(hiddenLayerSize);

%Setup Division of Data for Training, Validation, Testing

net.divideFcn = 'dividerand'; % Divide data randomly

net.divideMode = 'sample'; % Divide up every sample

net.divideParam.trainRatio = 50/100;

net.divideParam.valRatio = 25/100;

net.divideParam.testRatio = 25/100;

%Choose a Performance Function

%For a list of all performance functions type: help nnperformance

net.performFcn = 'crossentropy'; % Cross-Entropy

%Choose Plot Functions

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...

'plotconfusion', 'plotroc'};

%Train the Network

[net,tr] = train(net,x,t);

%Test the Network

y = net(x);

e = gsubtract(t,y);

performance = perform(net,t,y);

tind = vec2ind(t);

yind = vec2ind(y);

percentErrors = sum(tind ~= yind)/numel(tind);

%Recalculate Training, Validation and Test Performance

trainTargets = t .\* tr.trainMask{1};

valTargets = t .\* tr.valMask{1};

testTargets = t .\* tr.testMask{1};

trainPerformance = perform(net,trainTargets,y)

valPerformance = perform(net,valTargets,y)

testPerformance = perform(net,testTargets,y)

%figure, plotconfusion(t,y)

strcat(output1,'.fig')

saveas(plotconfusion(t,y),strcat(output1,'.fig'))

genFunction(net,'mynn.m');

%% Section 3. CellCognize testing of standard-mixed communities.

% In a first proof-of-concept experiment, we cultured E. coli MG1655, P. veronii and A. johnsonii individually to stationary phase, diluted cultures 1:1000 in PBS, and measured cells by FCM after staining with Sybr Green I either individually, or in different mixtures of all three strains combined. Individual and mixture data sets were analyzed with CellCognize using a set of five replicate ANN-5 classifiers, comparing expected added cell numbers of each of the three strains with their assigned class attributions from the ANN-5 classifiers.

% 3.1 Preparing a limited ANN with five standards only.

% data import. Read filtered, log-transformed and gated E. coli, P. veronii and A. johnsonii data files. This has five datasets

% path: /Files\_for\_Zenodo/FCM\_files/MIX\_experiment\_ACL\_AJH\_PVR/ filtered\_standards\_AJH\_ECL\_PVE.mat

%filtered\_standards={AJH1,ECL1,ECL2,PVR1,PVR2};

% Continue training, validating and testing ANN-5 with subsampled data set; here 5000 events per sample; as in sections 2.1 and 2.2

% Example output classifier file saved as /Files\_for\_Zenodo/NN\_file\_example/NNfunction\_wide\_anchor\_normz\_filtered\_trimix\_310819.m

%

% 3.2 Analyze the synthetic community mixtures.

%

%Read in and treat data files of either cultures alone, or in combinations. Same folder: /Files\_for\_Zenodo/FCM\_files/MIX\_experiment\_ACL\_AJH\_PVR

%as example: combination of 30/10/10 ECL/AJH/PVE

if strcmp(comm(end-3:end),'.fcs')

SP1= fca\_readfcs(comm);

else

a=readtable(comm);

SP1=table2array(a);

SP1=readtable(comm);% TODO change for the community file

%SP1=fca\_readfcs('t0\_A\_A-B\_SGPI\_repl1.fcs');

SP1=table2array(SP1);

size(SP1)

feat

scales1= SP1(:,feat(1)); %FSC-H

scales2= SP1(:,feat(2)); %SSC-H

if feat(3)==0

scales3=SP1(:,feat(6));

scales3(scales3>=0)=200;

else

scales3= SP1(:,feat(3));

end

%scales3= SP1(:,feat(3)); %FITC-H

scales4= SP1(:,feat(4)); %FSC-A

scales5= SP1(:,feat(5)); %SSC-A

scales6= SP1(:,feat(6)); %FITC-A

if feat(7)==0

scales7= SP1(:,feat(6)); %Width

scales7(scales7>=0)=20;

else

scales7= SP1(:,feat(7)); %Width

end

comFile= horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

input\_community=comFile;

size(comFile)

%community=comFile

community=comFile(comFile(:,1)<maxdata1,:);

size(community)

community=community(community(:,1)>=mindata1,:);

size(community)

community=community(community(:,2)<maxdata2,:);

size(community)

community=community(community(:,2)>mindata2,:);

size(community)

community=community(community(:,3)<maxdata3,:);

size(community)

community=community(community(:,3)>=mindata3,:);

size(community)

feat

community=community(community(:,4)<maxdata4,:);

size(community)

community=community(community(:,4)>=mindata4,:);

size(community)

community=community(community(:,5)<maxdata5,:);

size(community)

community=community(community(:,5)>=mindata5,:);

size(community)

community=community(community(:,6)<maxdata6,:);

size(community)

community=community(community(:,6)>=mindata6,:);

size(community)

community=community(community(:,7)<maxdata7,:);

size(community)

community=community(community(:,7)>=mindata7,:);

size(community)

%community=comFile;

%has 8 columns, remove column 8 check that for each file

scales1=log10(community(:,1));

scales2=log10(community(:,2));

scales3=log10(community(:,3));

scales4=log10(community(:,4));

scales5=log10(community(:,5));

scales6=log10(community(:,6));

scales7=log10(community(:,7));

input\_community=horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

%innput\_community=SP1

size(input\_community)

%add anchor line

%anchors=[2,2,2,2,2,2;6.6,6.6,5.7,6.3,6.3,6.0];

anchors=[2,2,2,2,2,2,1;6.6,6.6,5.7,6.3,6.3,6.0,3.3];

input\_community=vertcat(anchors,input\_community);

size(input\_community)

%transpose numeric array to be conform to NN input

input\_community=input\_community';

size(input\_community);

% run NN functions without thresholding

% vec2ind means that the index of the row is find where the value of 1 occurs. In this table it is the value that is rounded up to 1!

% NN functions are in switchdrive/final\_files/Community\_diversity\_analysis/NN functions

%make a table with the classes (1-5), a '0' (for the non-classified) - and one extra (33) that works as an anchor to fill the list properly (…)

classes=(0:3);

empty\_class=[3;0];

%NN function (1)

output\_restricted\_anchor1= net(input\_community); %available in example my nn

output\_restricted\_anchor\_final1=vec2ind(output\_restricted\_anchor1);

membership\_restricted1 = histc(output\_restricted\_anchor\_final1,unique(output\_restricted\_anchor\_final1));

classes\_restricted1=unique(output\_restricted\_anchor\_final1)

final\_community1= [classes\_restricted1;membership\_restricted1];

%now add the empty class to the final community

finalComm1=[final\_community1,empty\_class]

%apply a logical function to find corresponding values in the list with all categories in the file 'classes'

[lic,loc]=ismember(classes,finalComm1(1,:));

results1(2,lic)=finalComm1(2,loc(lic));

results1(1,:)=classes;

%produce final summary community and save as .csv. Modify the path if necessary.

final\_results=vertcat(results1)

T=array2table(final\_results','VariableNames',{'Class1','Count1'})

%write results to table, as example:

writetable(T,strcat(output1,'.csv'));% TODO

end

Matlab Cellcognize for communities

%% CellCognize%

% Script de base pour cellscanner communities

%

%

%% import references data into matrix

load('C:\Users\u0128864\Documents\KU\_LEUVEN\FLOWCYTOMETRY\Articles\CLEMENCE\DATA\CELLCOGNIZE\FCM\_files\FCM\_files\MIX\_experiment\_ACL\_AJH\_PVR\filtered\_standards\_AJH\_ECL\_PVE');

mindata1=100;

maxdata1=4000000;

mindata2=100;

maxdata2=4000000;

mindata3=100;

maxdata3=500000;

mindata4=100;

maxdata4=2000000;

mindata5=100;

maxdata5=2000000;

mindata6=100;

maxdata6=1000000;

mindata7=10;

maxdata7=2000;

features

features= string(features);

%features=cell2mat(features)

feat=zeros(1,7);

%feat=[find(features=='FSC-H'),find(features=='SSC-H'),find(features=='FITC-H'),find(features=='FSC-A'),find(features=='SSC-A'),find(features=='FITC-H'),find(features=='Width')];

feat(1,1)=find(features=='FSC-H');

feat(1,2)=find(features=='SSC-H');

feat(1,4)=find(features=='FSC-A');

feat(1,5)=find(features=='SSC-A');

if ~isempty(find(features=='FITC-H'))

feat(3)=find(features=='FITC-H');

feat(6)=find(features=='FITC-A');

elseif isempty(find(features=='FL1-H'))

feat(3)=0;

feat(6)=find(features=='FITC-A');

else

feat(3)=find(features=='FL1-H');

feat(6)=find(features=='FL1-A');

end

if ~isempty(find(features=='Width'))

feat(7)=find(features=='Width');

else

feat(7)=0

end

%% Data annotation:

%Gate the different standards to either individual or multiple subpopulations, depending on the 2D plot aspect of FCS-H vs SSC-H vs FITC-H (example in Fig. S1). Bead standards do not require further gating.

%Rename the bead standards

%species1=standard{1};

%species2=standard{2};

%Now combine the filtered gated standard files into one. This is the 32-standard input file for the ANN model.

%filtered\_standards={species1,species2};

%% Section 2. Artificial neural network reconstruction.

% The filtered and gated data sets of the standards (n ~ 3 × 105 to 1.5 × 106 events per standard) were used as input for the development of ANN models. The datasets were randomly subsampled to 10,000 events per standard using the datasample function (Matlab v. R2017a). Crucially, the lower and upper boundary thresholds imposed during the filtering process for each of the seven FCM parameters were added as two data points (“anchors”) per parameter to the first subsampled standard. This process of ‘anchoring’ was essential for the subsequent machine-learning algorithm. Subsampled anchored datasets were concatenated and used as input into the ANN model, during which they were further scaled (between ?1 and 1, hence the necessity to add the anchors) and randomly divided using Dividerand (Matlab v. R2017a ) into three blocks: a training set (50% of the data), a validation set (25%) and a testing set (25%).

% The ANN architecture consisted of a feed-forward back-propagation algorithm with one input, one hidden and one output layer. The input layer contained 7 nodes (corresponding to the 7 FCM parameters), whereas the output layer contained 5 (for the preliminary three-strain experiment) or 32 nodes (one for each of the standards in the full set). Input nodes were connected to the hidden layer by the sigmoid function (Matlab v. 2017a), whereas the hidden layer nodes (20) were connected to the output by the softmax transfer function (Matlab v. 2017a). The input matrix was trained using the trainscg function (Matlab v. 2017a) in a 1000-cycle of training, validation and testing (performance goal = 0 | time = Inf | min grad = 10-6 | max fail = 6). Performance of the ANN was evaluated by crossentropy. The outcome of the ANN model is a learned linear equation, termed the ANN classifier, describing the correlations between input parameters and the five (proof-of-concept experiment, ANN-5) or 32 classes of the standard dataset (ANN-32). The process of subsampling, anchoring, concatenation and training was repeated five times independently on the full datasets, generating five slightly different ANN classifiers. The performance of the ANN classifiers was assessed on the basis of confusion matrices (Matlab v. 2017a), representing predicted versus actual events for the complete in silico mixed set of standards, and the false prediction rate (as shown in Fig. S2).

%2.1. Subsampling and anchoring

%Data were first randomly subsampled to same number of events. File array name from previous scaling was 'filtered\_standards'.

%path /Files\_for\_Zenodo/FCM\_files

standard\_normz=[];

sample\_size=5000;% training siza, 20000 will not be enough so we selected 5000

for i=1:length(filtered\_standards)

standard\_normz{i,1}=datasample(filtered\_standards{1,i},sample\_size,1);

end;

standard\_normz=standard\_normz';

%%

%alternatively: subsample to n = 5000. This file is saved as 'standard\_normz\_restricted\_32.mat' for section 3.7

%Add the line with the anchors for proper and consistent scaling throughout all data sets.

% Do not no how to change that if I need to. 7x7

anchors=[2,2,2,2,2,2,1;6.6,6.6,5.7,6.3,6.3,6.0,3.3];

%anchors=[2,2,2,2,2,2;6.6,6.6,5.7,6.3,6.3,6.0];

standard\_normz{1}=vertcat(anchors,standard\_normz{1});

%%

%2.2. ANN selection, training and validation

%Continue ANN with subsampled data set (standard\_normz); here 5000 events per standard.

file\_length\_final=cellfun(@length, standard\_normz);

%%

file\_size=file\_length\_final;

file\_length\_final=[1 file\_length\_final];

file\_length\_final=cumsum(file\_length\_final);

input=vertcat(standard\_normz{:});

output=zeros(length(input),length(standard\_normz));

for i=1:length(file\_length\_final)-1

output(file\_length\_final(i):file\_length\_final(i+1)-1,i)=1;

end

input=input';

output=output';

x = input;

t = output;

%%

%Choose a Training Function

trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.

%Create a Pattern Recognition Network

hiddenLayerSize = 20;

net = patternnet(hiddenLayerSize);

%Setup Division of Data for Training, Validation, Testing

net.divideFcn = 'dividerand'; % Divide data randomly

net.divideMode = 'sample'; % Divide up every sample

net.divideParam.trainRatio = 50/100;

net.divideParam.valRatio = 25/100;

net.divideParam.testRatio = 25/100;

%Choose a Performance Function

%For a list of all performance functions type: help nnperformance

net.performFcn = 'crossentropy'; % Cross-Entropy

%Choose Plot Functions

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...

'plotconfusion', 'plotroc'};

%Train the Network

[net,tr] = train(net,x,t);

%Test the Network

y = net(x);

e = gsubtract(t,y);

performance = perform(net,t,y);

tind = vec2ind(t);

yind = vec2ind(y);

percentErrors = sum(tind ~= yind)/numel(tind);

%Recalculate Training, Validation and Test Performance

trainTargets = t .\* tr.trainMask{1};

valTargets = t .\* tr.valMask{1};

testTargets = t .\* tr.testMask{1};

trainPerformance = perform(net,trainTargets,y)

valPerformance = perform(net,valTargets,y)

testPerformance = perform(net,testTargets,y)

%figure, plotconfusion(t,y)

strcat(output1,'.fig')

saveas(plotconfusion(t,y),strcat(output1,'.fig'))

genFunction(net,'mynn.m');

%% Section 3. CellCognize testing of standard-mixed communities.

% In a first proof-of-concept experiment, we cultured E. coli MG1655, P. veronii and A. johnsonii individually to stationary phase, diluted cultures 1:1000 in PBS, and measured cells by FCM after staining with Sybr Green I either individually, or in different mixtures of all three strains combined. Individual and mixture data sets were analyzed with CellCognize using a set of five replicate ANN-5 classifiers, comparing expected added cell numbers of each of the three strains with their assigned class attributions from the ANN-5 classifiers.

% 3.1 Preparing a limited ANN with five standards only.

% data import. Read filtered, log-transformed and gated E. coli, P. veronii and A. johnsonii data files. This has five datasets

% path: /Files\_for\_Zenodo/FCM\_files/MIX\_experiment\_ACL\_AJH\_PVR/ filtered\_standards\_AJH\_ECL\_PVE.mat

%filtered\_standards={AJH1,ECL1,ECL2,PVR1,PVR2};

% Continue training, validating and testing ANN-5 with subsampled data set; here 5000 events per sample; as in sections 2.1 and 2.2

% Example output classifier file saved as /Files\_for\_Zenodo/NN\_file\_example/NNfunction\_wide\_anchor\_normz\_filtered\_trimix\_310819.m

%

% 3.2 Analyze the synthetic community mixtures.

%

%Read in and treat data files of either cultures alone, or in combinations. Same folder: /Files\_for\_Zenodo/FCM\_files/MIX\_experiment\_ACL\_AJH\_PVR

%as example: combination of 30/10/10 ECL/AJH/PVE

SP1=readtable(comm);% TODO change for the community file

%SP1=fca\_readfcs('t0\_A\_A-B\_SGPI\_repl1.fcs');

SP1=table2array(SP1);

size(SP1)

scales1= SP1(:,feat(1)); %FSC-H

scales2= SP1(:,feat(2)); %SSC-H

if feat(3)==0

scales3=SP1(:,feat(6));

scales3(scales3>=0)=200;

else

scales3= SP1(:,feat(3));

end

%scales3= SP1(:,feat(3)); %FITC-H

scales4= SP1(:,feat(4)); %FSC-A

scales5= SP1(:,feat(5)); %SSC-A

scales6= SP1(:,feat(6)); %FITC-A

if feat(7)==0

scales7= SP1(:,feat(6)); %Width

scales7(scales7>=0)=20;

else

scales7= SP1(:,feat(7)); %Width

end

comFile= horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

input\_community=comFile;

size(comFile)

%community=comFile

community=comFile(comFile(:,1)<maxdata1,:);

community=community(community(:,1)>mindata1,:);

community=community(community(:,2)<maxdata2,:);

community=community(community(:,2)>mindata2,:);

community=community(community(:,3)<maxdata3,:);

community=community(community(:,3)>mindata3,:);

ommunity=community(community(:,4)<maxdata4,:);

community=community(community(:,4)>mindata4,:);

community=community(community(:,5)<maxdata5,:);

community=community(community(:,5)>mindata5,:);

community=community(community(:,6)<maxdata6,:);

community=community(community(:,6)>mindata6,:);

community=community(community(:,7)<maxdata7,:);

community=community(community(:,7)>mindata7,:);

size(community)

%community=comFile;

%has 8 columns, remove column 8 check that for each file

scales1=log10(community(:,1));

scales2=log10(community(:,2));

scales3=log10(community(:,3));

scales4=log10(community(:,4));

scales5=log10(community(:,5));

scales6=log10(community(:,6));

scales7=log10(community(:,7));

%input\_community=horzcat(scales1,scales2,scales3,scales4,scales5,scales6,scales7);

innput\_community=SP1

size(input\_community)

%add anchor line

%anchors=[2,2,2,2,2,2;6.6,6.6,5.7,6.3,6.3,6.0];

anchors=[2,2,2,2,2,2,1;6.6,6.6,5.7,6.3,6.3,6.0,3.3];

input\_community=vertcat(anchors,input\_community);

size(input\_community)

%transpose numeric array to be conform to NN input

input\_community=input\_community';

size(input\_community);

% run NN functions without thresholding

% vec2ind means that the index of the row is find where the value of 1 occurs. In this table it is the value that is rounded up to 1!

% NN functions are in switchdrive/final\_files/Community\_diversity\_analysis/NN functions

%make a table with the classes (1-5), a '0' (for the non-classified) - and one extra (33) that works as an anchor to fill the list properly (…)

classes=(0:6);

empty\_class=[6;0];

%NN function (1)

output\_restricted\_anchor1= net(input\_community); %available in example my nn

output\_restricted\_anchor\_final1=vec2ind(output\_restricted\_anchor1)

membership\_restricted1 = histc(output\_restricted\_anchor\_final1,unique(output\_restricted\_anchor\_final1));

classes\_restricted1=unique(output\_restricted\_anchor\_final1)

final\_community1= [classes\_restricted1;membership\_restricted1];

%now add the empty class to the final community

finalComm1=[final\_community1,empty\_class]

%apply a logical function to find corresponding values in the list with all categories in the file 'classes'

[lic,loc]=ismember(classes,finalComm1(1,:));

results1(2,lic)=finalComm1(2,loc(lic));

results1(1,:)=classes;

%produce final summary community and save as .csv. Modify the path if necessary.

final\_results=vertcat(results1)

T=array2table(final\_results','VariableNames',{'Class1','Count1'})

%write results to table, as example:

writetable(T,strcat(output1,'.csv'));% TODO

end

mindata1=100;

maxdata1=4000000;

mindata2=100;

maxdata2=4000000;

mindata3=100;

maxdata3=500000;

mindata4=100;

maxdata4=2000000;

mindata5=100;

maxdata5=2000000;

mindata6=100;

maxdata6=1000000;

mindata7=10;

maxdata7=2000;

mindata1=0;

maxdata1=400000000000;

mindata2=0;

maxdata2=400000000000;

mindata3=0;

maxdata3=50000000000;

mindata4=0;

maxdata4=200000000000;

mindata5=0;

maxdata5=200000000000;

mindata6=0;

maxdata6=100000000000;

mindata7=0;

maxdata7=2000;

phenoflow for in\_silicocomm

#-\*- coding: utf-8 -\*-  
*"""  
Created on Tue Mar 8 11:13:05 2016  
  
@author: prubbens  
"""*#  
##############################################################################  
###Import packages############################################################  
##############################################################################  
#  
''' Imported packages for python '''  
import numpy as np  
import pandas as pd  
import pylab as plt  
import time  
import warnings  
import time  
import fcsparser  
from create\_community import \*  
from os import listdir  
#  
''' Imported packages from scikit-learn '''  
from sklearn.model\_selection import train\_test\_split #cross\_validation  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  
from sklearn import metrics  
from sklearn.ensemble import RandomForestClassifier  
#  
''' Define plotting style, ignore future warnings, start stopwatch '''  
plt.style.use('ggplot')  
warnings.simplefilter(action="ignore", category=FutureWarning) #Do not display futurewarnings  
start\_time = time.time() #Start stopwatch to determine runtime  
#  
##############################################################################  
###Read-in metadata and define global variables###############################  
##############################################################################  
#  
#  
'''  
Read-in metadata:  
This data is our expected outcome concerning the in vitro communities in the abundance gradient  
'''  
  
#  
##############################################################################  
###Functions##################################################################  
##############################################################################  
#  
#  
''' Return path of microbial community of interest '''  
''' Input: '''  
'comb == 0: Pseudomonas putida -- Pseudomonas fluorescens (initial low performance)'  
'comb == 1: Agrobacter rhizogenes -- Janthinobacterium sp. B3 (initial medium performance)'  
'comb == 2: Shewanella oneidensis -- Micrococcus luteus (initial high performance)'  
#  
#  
def get\_path(comb):  
 if comb == 0:  
 path\_insilico = 'PC\_filtered\_1\_11/'  
 path\_invitro = '1\_11\_filtered/'  
 nrep\_insilico = 3  
 print(path\_invitro)  
 elif comb == 1:  
 path\_insilico = 'PC\_filtered\_rerun\_16\_25/'  
 path\_invitro = '16\_25\_rerun\_filtered/'  
 nrep\_insilico = 4  
 else:  
 path\_insilico = 'PC\_filtered\_3\_17/'  
 path\_invitro = '3\_17\_filtered/'  
 nrep\_insilico = 3  
 nrep\_invitro = 3  
 return path\_insilico, nrep\_insilico, path\_invitro, nrep\_invitro  
#  
#  
''' Filter out features you don't want to use for your classifier '''  
'Input: pandas dataframe'  
'Output: list of features'  
#  
#  
def get\_features(df):  
 features = list(df.columns)  
 if (len(features) == 15):  
 features.remove('species')  
 features.remove('Time')  
 features.remove('Width')  
 return features  
#  
#  
''' Return in silico community containing two bacterial populations '''  
'datalist: list of filenames containing individual bacterial populations'  
'n\_subsample: number of cells to sample per bacterial population'  
'path: path to directory'  
'nrep: number of replicates'  
#  
#  
def get\_insilico\_comm(datalist, n\_subsample, path, nrep):  
 df0 = pd.DataFrame()  
 df1 = pd.DataFrame()  
 for i in np.arange(0, nrep):  
 if datalist[i][-4:] == '.csv':  
 df\_singlespecies = pd.read\_csv(path + datalist[i], header=0,index\_col=False)  
 elif datalist[i][-4:] == '.fcs':  
 meta, df\_singlespecies = fcsparser.parse(path + datalist[i], reformat\_meta=True)  
 df\_singlespecies['species'] = 0  
 df0 = pd.concat([df0, df\_singlespecies], axis=0, ignore\_index=True)  
 df0 = df0.sample(n\_subsample, random\_state=903, replace=False)  
 for j in np.arange(nrep, 2 \* nrep):  
 if datalist[i][-4:] == '.csv':  
 df\_singlespecies = pd.read\_csv(path + datalist[j], header=0,index\_col=False)  
 elif datalist[i][-4:] == '.fcs':  
 meta, df\_singlespecies = fcsparser.parse(path + datalist[j], reformat\_meta=True)  
 df\_singlespecies['species'] = 1  
 df1 = pd.concat([df1, df\_singlespecies], axis=0, ignore\_index=True)  
 df1 = df1.sample(n\_subsample, random\_state=1503, replace=False)  
 df = pd.concat([df0, df1], axis=0, ignore\_index=True)  
 return df  
#  
#  
''' Return in silico community containing two bacterial populations in varying abundances '''  
'datalist: list of filenames containing individual bacterial populations'  
'n\_sample: number of cells to sample per bacterial population'  
'abun: relative abundance (between 0 and 1) of first bacterial population'  
'path: path to directory'  
'nrep: number of replicates'  
#  
#  
def get\_insilico\_comm\_abun(datalist, n\_sample, abun, path, nrep):  
 df0 = pd.DataFrame()  
 df1 = pd.DataFrame()  
 for i in np.arange(0, nrep):  
 if datalist[i][-4:]=='.csv':  
 df\_singlespecies = pd.read\_csv(path + datalist[i], header=0,index\_col=False)  
 elif datalist[i][-4:]=='.fcs':  
 meta, df\_singlespecies = fcsparser.parse(path+datalist[i], reformat\_meta=True)  
  
 df\_singlespecies['species'] = 0  
 df0 = pd.concat([df0, df\_singlespecies], axis=0, ignore\_index=True)  
 df0 = df0.sample(int(n\_sample \* abun), random\_state=27, replace=False)  
 for j in np.arange(nrep, 2 \* nrep):  
 if datalist[i][-4:]=='.csv':  
 df\_singlespecies = pd.read\_csv(path + datalist[j], header=0,index\_col=False)  
 elif datalist[i][-4:]=='.fcs':  
 meta, df\_singlespecies = fcsparser.parse(path+datalist[j], reformat\_meta=True)  
  
 df\_singlespecies['species'] = 1  
 df1 = pd.concat([df1, df\_singlespecies], axis=0, ignore\_index=True)  
 df1 = df1.sample(int(n\_sample \* (1. - abun)), random\_state=633, replace=False)  
 df = pd.concat([df0, df1], axis=0, ignore\_index=True)  
 return df  
#  
#  
''' Sample cells of synthetic bacterial community '''  
'idx: 0 or 1 (first or second bacterial population)'  
'datalist: list of filenames containing synthetic communities'  
'n\_sample: number of cells to sample per invitro community'  
'path: path to directory'  
'nrep: number of replicates'  
#  
#  
def get\_invitro\_comm(idx, datalist, n\_sample, path, nrep):  
 df = pd.DataFrame()  
 for i in np.arange(int(idx \* nrep), int((idx + 1) \* nrep)):  
 if datalist[i][-4:]=='.csv':  
 df\_rep = pd.read\_csv(path + datalist[i], header=0,index\_col=False)  
 df = pd.concat([df, df\_rep], axis=0, ignore\_index=True)  
 elif datalist[i][-4:]=='.fcs':  
 meta, df\_rep = fcsparser.parse(path+datalist[i], reformat\_meta=True)  
 df = pd.concat([df, df\_rep], axis=0, ignore\_index=True)  
  
 if (df.shape[0] > n\_sample):  
 df = df.sample(n\_sample, random\_state=5495, replace=False)  
 print(df)  
 return df  
#  
#  
''' Calculate relative abundance p and alpha diversity parameters D1 and D2 '''  
'cluster: array of (predicted) cell labels'  
'n\_clust: number of different clusters'  
#  
#  
def calc\_D1\_D2(cluster, n\_clust):  
 cluster = pd.DataFrame(cluster)  
 cluster.columns = ['clust']  
 pp = 0  
 p\_d1 = 0.  
 p\_d2 = 0.  
 min\_clust = np.amin(cluster.clust)  
 max\_clust = np.amax(cluster.clust)  
 for i in np.arange(min\_clust, max\_clust + 1):  
 p = np.float64(cluster.loc[cluster['clust'] == i].shape[0] / cluster.shape[0])  
 p\_d1 += p \* np.log(p)  
 p\_d2 += p \*\* 2.  
 if pp == 0.:  
 pp = p  
 return pp, np.exp(-1. \* p\_d1), 1. / p\_d2  
#  
#  
''' Split in silico community into a training and validation/test set '''  
'Input: dataframe'  
#  
#  
def get\_train\_test(df):  
 features = list(df.columns)  
 art\_x\_train, art\_x\_test, art\_y\_train, art\_y\_test = train\_test\_split(df[features[0:12]], df['species'],  
 test\_size=0.3, random\_state=588)  
 return art\_x\_train, art\_x\_test, art\_y\_train, art\_y\_test  
#  
#  
''' Train Linear Discriminant analysis on training set and evaluate on test set '''  
'x\_train: dataframe training set'  
'y\_train: labels of training set'  
'x\_test: dataframe test set'  
#  
#  
def perform\_lda(x\_train, x\_test, y\_train):  
 lda = LinearDiscriminantAnalysis(solver='lsqr')  
 lda.fit(x\_train, y\_train)  
 return lda.predict(x\_test)  
#  
#  
''' Perform LDA with dimensionality reduction '''  
#  
#  
def perform\_lda\_dimred(x\_train, x\_test, y\_train):  
 lda = LinearDiscriminantAnalysis(solver='svd')  
 lda.fit(x\_train, y\_train)  
 return lda.transform(x\_test)  
#  
#  
''' Plot feature importances from Random Forest classifier '''  
'features: list of features'  
'feature importances: RF.feature\_importances\_'  
#  
#  
def plot\_feature\_importances(features, feature\_importances):  
 df = pd.DataFrame(feature\_importances, index=features)  
 df.sort(axis=1, ascending=False, inplace=True)  
 df.columns = ['feature\_importances']  
 pos = np.arange(0, len(features)) + 0.5  
 plt.figure(figsize=(20, 12))  
 plt.barh(pos, df.feature\_importances, color='darkorange', align='center')  
 plt.yticks(pos, df.index)  
 plt.xlabel('Importance')  
 plt.title('Feature Importances')  
 plt.axis([0, 0.25, 0, 12])  
 plt.show()  
 plt.savefig('RF\_featureimportances\_2Species\_3.png')  
#  
#  
''' Train using Linear Discriminant Analysis on training set and return classifier '''  
'x\_train: dataframe training set'  
'y\_train: labels of training set'  
#  
#  
def return\_lda\_class(x\_train, y\_train):  
 lda = LinearDiscriminantAnalysis(solver='svd')  
 lda.fit(x\_train, y\_train)  
 return lda  
#  
#  
''' Train Random Forest classifier on training set and return classifier '''  
'x\_train: dataframe training set'  
'y\_train: labels of training set'  
#  
#  
def return\_RF\_class(x\_train, y\_train):  
 rf = RandomForestClassifier(n\_estimators=200, criterion='gini', random\_state=6)  
 rf.fit(x\_train, y\_train)  
 return rf  
#  
#  
''' Train Linear Discriminant analysis on training set and evaluate (labels) on test set '''  
'x\_train: dataframe training set'  
'y\_train: labels of training set'  
'x\_test: dataframe test set'  
#  
#  
def perform\_RF(x\_train, x\_test, y\_train):  
 rf = RandomForestClassifier(n\_estimators=200, criterion='gini', random\_state=3)  
 rf.fit(x\_train, y\_train)  
 plot\_feature\_importances(features, rf.feature\_importances\_)  
 return rf.predict(x\_test)  
#  
#  
''' Train Linear Discriminant analysis on training set and evaluate (probabilities) on test set '''  
'x\_train: dataframe training set'  
'y\_train: labels of training set'  
'x\_test: dataframe test set'  
#  
#  
def perform\_RF\_scores(x\_train, x\_test, y\_train):  
 rf = RandomForestClassifier(n\_estimators=200, criterion='gini', random\_state=3)  
 rf.fit(x\_train, y\_train)  
 std\_fi = np.std([tree.feature\_importances\_ for tree in rf.estimators\_], axis=0)  
 plot\_feature\_importances(features, rf.feature\_importances\_, std\_fi)  
 return rf.predict\_proba(x\_test)[:, 1]  
#  
#  
''' Create in silico abundance gradient and analyze it using classifier trained on initial in silico community '''  
'datalist: list of filenames containing individual bacterial populations'  
'Nax: number of cells to sample per population in in silico community'  
'Nabun: total number of cells to sample for in silico communities making up an abundance gradient'  
'path\_insilico: path to directory containing in silico communities'  
'nrep\_insilico: number of replictates'  
#  
#  
def perform\_insilico\_analysis(datalist\_insilico, Nax, Nabun, path\_insilico, nrep\_insilico):  
 df\_insilico = get\_insilico\_comm(datalist\_insilico, Nax, path\_insilico, nrep\_insilico)  
 features = get\_features(df\_insilico)  
 features = get\_features(df\_insilico)  
 #clf = return\_RF\_class(df\_insilico[features], df\_insilico.species)  
 clf = return\_lda\_class(df\_insilico[features], df\_insilico.species)  
 percentages = [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99]  
 noc = len(percentages)  
 p = np.zeros(noc)  
 D1 = np.zeros(noc)  
 D2 = np.zeros(noc)  
 dummy = 0  
 for pct in percentages:  
 df\_insilico\_abun = get\_insilico\_comm\_abun(datalist\_insilico, Nabun, pct, path\_insilico, nrep\_insilico)  
 pred = clf.predict(df\_insilico\_abun[features])  
 p[dummy], D1[dummy], D2[dummy] = calc\_D1\_D2(pred, 2)  
 dfresult = pd.DataFrame(percentages, columns=['Theoretical Abundances'])  
 dummy += 1  
 dfresult['D1'] = D1  
 dfresult['D2'] = D2  
 dfresult['p0'] = p  
 dfresult.sort(columns='Theoretical Abundances', inplace=True)  
 dfresult.to\_csv('dfinsilico\_abun.csv')  
 return dfresult  
#  
#  
''' Create in silico abundance gradient and analyze it using classifier trained on initial in silico community '''  
'datalist\_insilico: list of filenames containing individual bacterial populations'  
'datalist\_invitro: list of filenames containing synthetic bacterial communities in varying abundances'  
'Nax: number of cells to sample per population in in silico community'  
'Ninvitro: total number of cells to sample for in vitro communities making up an abundance gradient'  
'path\_insilico: path to directory containing in silico communities'  
'path\_invitro: path to directory containing in silico communities'  
'nrep\_insilico: number of replictates of individual bacterial populations'  
'nrep\_invitro: number of replictates of in vitro communities'  
'noc: number of communities making up an abundance gradient (13 in the paper)'  
'targetabundances: metadata containing outcome (in vitro created) abundances'  
#  
#  
def perform\_invitro\_analysis(datalist\_insilico, datalist\_invitro, Nax, Ninvitro, path\_insilico, path\_invitro,  
 nrep\_insilico, nrep\_invitro, noc, features,output='', targetabundances=''):  
 df\_insilico = get\_insilico\_comm(datalist\_insilico, Nax, path\_insilico, nrep\_insilico)  
 features = features#get\_features  
 print('DF')  
 print(df\_insilico)  
 print(df\_insilico.species)  
 clf = return\_RF\_class(df\_insilico[features], df\_insilico.species)  
 theor\_abundances = [50]#[10, 1, 20, 30, 40, 50, 5, 60, 70, 80, 90, 95, 99]  
 p = np.zeros(noc)  
 D1 = np.zeros(noc)  
 D2 = np.zeros(noc)  
 D3 = np.zeros(noc)  
 D4 = np.zeros(noc)  
 D5 = np.zeros(noc)  
 N = np.zeros(noc)  
 for i in np.arange(0, noc):  
 df\_invitro = get\_invitro\_comm(i, datalist\_invitro, Ninvitro, path\_invitro, nrep\_invitro)  
 print(df\_invitro)  
 pred = clf.predict(df\_invitro[features])  
 print(pred)  
 #p[i], D1[i], D2[i] = calc\_D1\_D2(pred, 2)  
 D1[i]=np.count\_nonzero(pred==0)  
 D2[i] = np.count\_nonzero(pred == 1)  
 D3[i]=np.count\_nonzero(pred==2)  
 D4[i] = np.count\_nonzero(pred == 3)  
 D5[i] = np.count\_nonzero(pred == 4)  
 p[i]=D1[i]/D2[i]  
 N[i] = df\_invitro.shape[0]  
 dfresult = pd.DataFrame(theor\_abundances, columns=['Theoretical Abundances'])  
 dfresult['p0'] = p  
 dfresult['D1'] = D1  
 dfresult['D2'] = D2  
 dfresult['D3']= D3  
 dfresult['D4'] = D4  
 dfresult['D5'] = D5  
 dfresult['N'] = N  
 #dfresult.sort(columns='Theoretical Abundances', inplace=True)  
 #dfresult['Target abundances'] = targetabundances  
 print(dfresult)  
 dfresult.to\_csv(output)  
 return dfresult  
#  
#  
##############################################################################  
###Call functions#############################################################  
##############################################################################  
#targetabundances = pd.read\_excel('targetabundances.xlsx', index\_col='File')  
#  
''' Global variables'''  
  
def phenoflow(datalist\_insilico,datalist\_invitro,features, output):  
 Nax = 5000  
 Ncomm = 2000

invitro\_combination = 2 # 0: low, 1: medium, 2: high  
#  
#path\_insilico, nrep\_insilico, path\_invitro, nrep\_invitro = get\_path(invitro\_combination)  
 nrep\_insilico= 1  
 nrep\_invitro= 1  
 path\_insilico=''  
 path\_invitro=''  
  
#datalist\_insilico = sorted(listdir(path\_insilico))  
#datalist\_invitro = sorted(listdir(path\_invitro))  
#datalist\_insilico = ['C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/stelios\_fodelianakis/FlowRepository\_FR-FCM-ZYG6\_files/pure\_E111\_37C\_t2.fcs','C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/stelios\_fodelianakis/FlowRepository\_FR-FCM-ZYG6\_files/pure\_B42\_37C\_t2.fcs']  
#datalist\_invitro = ['C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/test.csv']  
#a='C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/stelios\_fodelianakis/FlowRepository\_FR-FCM-ZYG6\_files/pure\_E111\_37C\_t2.fcs'  
#b='C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/stelios\_fodelianakis/FlowRepository\_FR-FCM-ZYG6\_files/pure\_B42\_37C\_t2.fcs'  
 #df= insilico\_com([a,b],1000)  
 #saveTable(df,'C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/test.csv')  
 #datalist\_invitro=['C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/test.csv']  
 noc = int(len(datalist\_invitro)/nrep\_invitro)  
#df\_result\_insilico = perform\_insilico\_analysis(datalist\_insilico, Nax, Ncomm, path\_insilico, nrep\_insilico,output)  
 df\_result\_invitro = perform\_invitro\_analysis(datalist\_insilico, datalist\_invitro, Nax, Ncomm, path\_insilico, path\_invitro, nrep\_insilico, nrep\_invitro, noc,features,output )

In vitro community script

import runScript as s  
import os.path  
from os import listdir  
from os import path  
import matlab.engine  
from pheno\_prediction import \*  
  
eng = matlab.engine.start\_matlab()  
  
# Figure 1 = > evaluation pour in silico-communities  
  
D = "C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/"  
  
# 1 Jasmine : 2 sp, 3 time point = 6 fichier x 2 (ajouter les contacts?)(ajouter les stains?)  
dossier1= "Jasmine Heyse/"  
references1=['t0\_A\_A-fresh\_SG\_repl1.fcs','t0\_A\_A-fresh\_SGPI\_repl1.fcs','t0\_B\_B-fresh\_SG\_repl1.fcs','t0\_B\_B-fresh\_SGPI\_repl1.fcs','t1\_A\_A-fresh\_SG\_repl1.fcs','t1\_A\_A-fresh\_SGPI\_repl1.fcs','t1\_B\_B-fresh\_SG\_repl1.fcs','t1\_B\_B-fresh\_SGPI\_repl1.fcs','t2\_A\_A-fresh\_SG\_repl1.fcs','t2\_A\_A-fresh\_SGPI\_repl1.fcs','t2\_B\_B-fresh\_SG\_repl1.fcs','t2\_B\_B-fresh\_SGPI\_repl1.fcs']  
references1=[D+dossier1+i for i in references1]  
predictions1=['t0\_A\_A-fresh\_SG\_repl2.fcs','t0\_A\_A-fresh\_SGPI\_repl2.fcs','t0\_B\_B-fresh\_SG\_repl2.fcs','t0\_B\_B-fresh\_SGPI\_repl2.fcs','t1\_A\_A-fresh\_SG\_repl2.fcs','t1\_A\_A-fresh\_SGPI\_repl2.fcs','t1\_B\_B-fresh\_SG\_repl2.fcs','t1\_B\_B-fresh\_SGPI\_repl2.fcs','t2\_A\_A-fresh\_SG\_repl2.fcs','t2\_A\_A-fresh\_SGPI\_repl2.fcs','t2\_B\_B-fresh\_SG\_repl2.fcs','t2\_B\_B-fresh\_SGPI\_repl2.fcs']  
predictions1=[D+dossier1+i for i in predictions1]  
species1=['A-0-SG','A-0-SGPI','B-0-SG','B-0-SGPI','A-1-SG','A-1-SGPI','B-1-SG','B-1-SGPI','A-2-SG','A-2-SGPI','B-2-SG','B-2-SGPI']  
#channels1=[ 'FSC-A','FSC-W','FSC-H','SSC-A','SSC-W','SSC-H','FL1-A','FITC-W','FITC-H','FL3-A','PerCP-Cy5.5-W','PerCP-Cy5.5-H','APC-A','APC-W','APC-H','AmCyan-A','AmCyan-W','AmCyan-H','APC-Cy7-A','APC-Cy7-W','APC-Cy7-H','dsRed-A','dsRed-W','dsRed-H','eCFP-A','eCFP-W','eCFP-H','FL2-A','PE-Cy7-W','PE-Cy7-H']  
channels1=["FSC-A","FSC-W","FSC-H","SSC-A","SSC-W","SSC-H","FITC-A","FITC-W","FITC-H","PerCP-Cy5.5-A","PerCP-Cy5.5-W","PerCP-Cy5.5-H","APC-A","APC-W","APC-H","AmCyan-A","AmCyan-W","AmCyan-H","APC-Cy7-A","APC-Cy7-W","APC-Cy7-H","dsRed-A","dsRed-W","dsRed-H","eCFP-A","eCFP-W","eCFP-H","PE-Cy7-A","PE-Cy7-W","PE-Cy7-H"]  
  
dic1={'FITC-A':'FL1-A','PE-Cy7-A':'FL2-A','PerCP-Cy5.5-A':'FL3-A'}  
  
# 2 Stelios : 3 sp t4 training t5 prediction (t1,t3,t5,t7) vs (t2,t4,t6,t8)?  
dossier2 = "stelios\_fodelianakis/FlowRepository\_FR-FCM-ZYG6\_files/"  
references2 = ['pure\_E111\_37C\_t1.fcs','pure\_E111\_37C\_t3.fcs','pure\_E111\_37C\_t5.fcs','pure\_E111\_37C\_t7.fcs','pure\_B42\_37C\_t1.fcs','pure\_B42\_37C\_t3.fcs','pure\_B42\_37C\_t5.fcs','pure\_B42\_37C\_t7\_10-1.fcs','pure\_E310\_37C\_t1.fcs','pure\_E310\_37C\_t3.fcs','pure\_E310\_37C\_t5.fcs','pure\_E310\_37C\_t7\_10-1.fcs']  
references2=[D+dossier2+i for i in references2]  
predictions2 = ['pure\_E111\_37C\_t2.fcs','pure\_E111\_37C\_t4.fcs','pure\_E111\_37C\_t6.fcs','pure\_E111\_37C\_t8\_10-1.fcs','pure\_B42\_37C\_t2.fcs','pure\_B42\_37C\_t4.fcs','pure\_B42\_37C\_t6\_10-1.fcs','pure\_B42\_37C\_t8\_10-1.fcs','pure\_E310\_37C\_t2.fcs','pure\_E310\_37C\_t4.fcs','pure\_E310\_37C\_t6.fcs','pure\_E310\_37C\_t8\_10-1.fcs']  
predictions2=[D+dossier2+i for i in predictions2]  
species2 = ['E111-1','E111-2','E111-3','E111-4','B41-1','B41-2','B41-3','B41-4','E310-1','E310-2','E310-3','E310-4']  
#channels2=['FSC-A','SSC-A','FL1-A','FL2-A','FL3-A','FL4-A','FSC-H','SSC-H','FL1-H','FL2-H','FL3-H','FL4-H','Width']  
channels2=["FSC-A","SSC-A","FL1-A","FL2-A","FL3-A","FL4-A","FSC-H","SSC-H","FL1-H","FL2-H","FL3-H","FL4-H",'Width']  
  
  
dic2={}  
  
# 3 Ji Youn Lee : human cells (HEK293, HUH7,U937, T98G) / (comparer avec les autres en enlevant les fluo channels oui?+ Jurkat, nb2violet, farre, thp1 ...) (nb3-11& Jurkat TPH1) with channels  
dossier3 = "Ji\_Youn\_Lee/FlowRepository\_FR-FCM-ZZUZ\_files/"  
references3 = ['C\_Mono\_Jurkat\_day2.fcs','C\_Mono\_THP1\_day2.fcs','Jurkat\_Mono\_Day2.fcs','THP1\_Mono\_Day2.fcs',  
 'Jurkat\_Violet\_mono\_Day2.fcs','THP1\_FarRed\_mono\_Day2.fcs','HEK239\_150206\_HEK293\_day2.fcs',  
 'HuH7\_150205\_HuH7\_day2.fcs','A SET\_Nb2-11\_violet\_day2.fcs','T98G\_MAR10-15\_1%\_T98G\_day2\_1%\_CFSE\_20000.fcs',  
 'U2OS\_150205\_U2OS\_day2.fcs','U937\_MAR5-10\_10%\_U937\_day2\_CFSE\_50000\_1.fcs']  
references3=[D+dossier3+i for i in references3]  
predictions3 = ['C\_Mono\_Jurkat\_day3.fcs','C\_Mono\_THP1\_day3.fcs','Jurkat\_Mono\_Day3.fcs','THP1\_Mono\_Day3.fcs','Jurkat\_Violet\_mono\_Day3.fcs','THP1\_FarRed\_mono\_Day3.fcs','HEK239\_150206\_HEK293\_day3.fcs','HuH7\_150205\_HuH7\_day3.fcs','A SET\_Nb2-11\_violet\_day3.fcs','T98G\_MAR10-15\_1%\_T98G\_day2\_1%\_CFSE\_20001.fcs','U2OS\_150205\_U2OS\_day3.fcs','U937\_MAR5-10\_10%\_U937\_day2\_CFSE\_50000\_2.fcs']  
predictions3=[D+dossier3+i for i in predictions3]  
species3 = ['J-1','THP1-1','J-2','THP1-2','J-V','THP1-R','HEK293','HuH7','Nb2','T98G','U2OS','U937']  
#channels3=["FSC-A","FSC-H","FL2-A","SSC-A","SSC-H","FL3-A","FL1-A"]  
channels3=["FSC-A","FSC-H","FSC-W","SSC-A","SSC-H","SSC-W","FITC-A"]  
  
dic3={'FITC-A':'FL1-A','FSC-W':'FL2-A','SSC-W':'FL3-A'}  
  
references3b = ['Jurkat\_Mono\_Day2.fcs','THP1\_Mono\_Day2.fcs']  
references3b=[D+dossier3+i for i in references3b]  
predictions3b = ['Jurkat\_Mono\_Day3.fcs','THP1\_Mono\_Day3.fcs']  
predictions3b=[D+dossier3+i for i in predictions3b]  
species3b = ['Jurkat','THP1']  
#channels3b=['FSC-A','FSC-W','FSC-H','SSC-A','SSC-W','SSC-H','FL1-A','FL2-A','FL3-A','PE-Cy7-A','APC-A','APC-Cy7-A']  
channels3b=["FSC-A","FSC-W","FSC-H","SSC-A","SSC-W","SSC-H","FITC-A","PE-A","PerCP-Cy5.5-A","PE-Cy7-A","APC-A","APC-Cy7-A"]  
  
dic3b={'FITC-A':'FL1-A','PE-A':'FL2-A','PerCP-Cy5.5-A':'FL3-A'}  
  
references3c = ['Jurkat\_Violet\_mono\_Day2.fcs','THP1\_FarRed\_mono\_Day2.fcs']  
references3c=[D+dossier3+i for i in references3c]  
predictions3c =['Jurkat\_Violet\_mono\_Day3.fcs','THP1\_FarRed\_mono\_Day3.fcs']  
predictions3c=[D+dossier3+i for i in predictions3c]  
species3c = ['Jurkat','THP1']  
#channels3c=['FSC-A','FSC-W','FSC-H','SSC-A','SSC-W','SSC-H','FL1-A','FL2-A','FL3-A','V450-A']  
channels3c=["FSC-A","FSC-W","FSC-H","SSC-A","SSC-W","SSC-H","FITC-A","PE-A","APC-A","V450-A"]  
  
dic3c={'FITC-A':'FL1-A','PE-A':'FL2-A','APC-A':'FL3-A'}  
  
# 4 Rubbens monocultures 20spc x2 x2 (accuri /FACS)  
dossier4 = "PHENOFLOW\_DATA/FlowRepository\_FR-FCM-ZZSH\_files/" #accurie  
dossier5= "PHENOFLOW\_DATA/FlowRepository\_FR-FCM-ZY6M\_files/" #FACS  
references4 = ['01\_rep1.fcs','02\_rep1.fcs','03\_rep1.fcs','04\_rep1.fcs','05\_rep1.fcs','08\_rep1.fcs','09\_rep1.fcs','10\_rep1.fcs','11\_rep1.fcs','12\_rep1.fcs','13\_rep1.fcs','15\_rep1.fcs','16\_rep1.fcs','17\_rep1.fcs','18\_rep1.fcs','19\_rep1.fcs','20\_rep1.fcs','21\_rep1.fcs','23\_rep1.fcs','25\_rep1.fcs']  
references4=[D+dossier4+i for i in references4]  
predictions4 = ['01\_rep2.fcs','02\_rep2.fcs','03\_rep2.fcs','04\_rep2.fcs','05\_rep2.fcs','08\_rep2.fcs','09\_rep2.fcs','10\_rep2.fcs','11\_rep2.fcs','12\_rep2.fcs','13\_rep2.fcs','15\_rep2.fcs','16\_rep2.fcs','17\_rep2.fcs','18\_rep2.fcs','19\_rep2.fcs','20\_rep2.fcs','21\_rep2.fcs','23\_rep2.fcs','25\_rep2.fcs']  
predictions4=[D+dossier4+i for i in predictions4]  
species4 = ['1','2','3','4','5','8','9','10','11','12','13','15','16','17','18','19','20','21','23','25']  
#channels4=['FSC-A','SSC-A','FL1-A','FL2-A','FL3-A','FL4-A','FSC-H','SSC-H','FL1-H','FL2-H','FL3-H','FL4-H','Width']  
channels4=["FSC-A","SSC-A","FL1-A","FL2-A","FL3-A","FL4-A","FSC-H","SSC-H","FL1-H","FL2-H","FL3-H","FL4-H",'Width']  
  
dic4={}  
  
references5 = ['01\_rep1.fcs','02\_rep1.fcs','03\_rep1.fcs','04\_rep1.fcs','05\_rep1.fcs','08\_rep1.fcs','09\_rep1.fcs','10\_rep1.fcs','11\_rep1.fcs','12\_rep1.fcs','13\_rep1.fcs','15\_rep1.fcs','16\_rep1.fcs','17\_rep1.fcs','18\_rep1.fcs','19\_rep1.fcs','20\_rep1.fcs','21\_rep1.fcs','23\_rep1.fcs','25\_rep1.fcs']  
references5=[D+dossier5+i for i in references5]  
predictions5 = ['01\_rep2.fcs','02\_rep2.fcs','03\_rep2.fcs','04\_rep2.fcs','05\_rep2.fcs','08\_rep2.fcs','09\_rep2.fcs','10\_rep2.fcs','11\_rep2.fcs','12\_rep2.fcs','13\_rep2.fcs','15\_rep2.fcs','16\_rep2.fcs','17\_rep2.fcs','18\_rep2.fcs','19\_rep2.fcs','20\_rep2.fcs','21\_rep2.fcs','23\_rep2.fcs','25\_rep2.fcs']  
predictions5=[D+dossier5+i for i in predictions5]  
species5 = ['1','2','3','4','5','8','9','10','11','12','13','15','16','17','18','19','20','21','23','25']  
#channels5=['FSC-A','FSC-W','FSC-H','SSC-A','SSC-W','SSC-H','FL1-A','FITC-W','FITC-H','FL2-A','PE-W','PE-H','FL3-A','PerCP-Cy5.5-W','PerCP-Cy5.5-H','PE-Cy7-A','PE-Cy7-W','PE-Cy7-H','APC-A','APC-W','APC-H','APC-Cy7-A','APC-Cy7-W','APC-Cy7-H','V450-A','V450-W','V450-H','V500-A','V500-W','V500-H']  
channels5=["FSC-A","FSC-W","FSC-H","SSC-A","SSC-W","SSC-H","FITC-A","FITC-W","FITC-H","PE-A","PE-W","PE-H","PerCP-Cy5.5-A","PerCP-Cy5.5-W","PerCP-Cy5.5-H","PE-Cy7-A","PE-Cy7-W","PE-Cy7-H","APC-A","APC-W","APC-H","APC-Cy7-A","APC-Cy7-W","APC-Cy7-H","V450-A","V450-W","V450-H","V500-A","V500-W","V500-H"]  
  
dic5={'FITC-A':'FL1-A','PE-A':'FL2-A','PerCP-Cy5.5-A':'FL3-A'}  
  
#Cellcognize 32 monocultures  
dossier6 = "CELLCOGNIZE/FCM\_files/for\_Clémence/Filtered/"  
references6 = ['1\_filtered.csv','2\_filtered.csv','3\_filtered.csv','4\_filtered.csv','5\_filtered.csv','6\_filtered.csv','7\_filtered.csv','8\_filtered.csv','9\_filtered.csv','10\_filtered.csv','11\_filtered.csv','12\_filtered.csv','13\_filtered.csv','14\_filtered.csv','15\_filtered.csv','16\_filtered.csv','17\_filtered.csv','18\_filtered.csv','19\_filtered.csv','20\_filtered.csv','21\_filtered.csv','22\_filtered.csv','23\_filtered.csv','24\_filtered.csv','25\_filtered.csv','26\_filtered.csv','27\_filtered.csv','28\_filtered.csv','29\_filtered.csv','30\_filtered.csv','31\_filtered.csv']  
references6 =[D+dossier6+i for i in references6]  
predictions6 = ['1\_filtered.csv','2\_filtered.csv','3\_filtered.csv','4\_filtered.csv','5\_filtered.csv','6\_filtered.csv','7\_filtered.csv','8\_filtered.csv','9\_filtered.csv','10\_filtered.csv','11\_filtered.csv','12\_filtered.csv','13\_filtered.csv','14\_filtered.csv','15\_filtered.csv','16\_filtered.csv','17\_filtered.csv','18\_filtered.csv','19\_filtered.csv','20\_filtered.csv','21\_filtered.csv','22\_filtered.csv','23\_filtered.csv','24\_filtered.csv','25\_filtered.csv','26\_filtered.csv','27\_filtered.csv','28\_filtered.csv','29\_filtered.csv','30\_filtered.csv','31\_filtered.csv']  
predictions6=[D+dossier6+i for i in predictions6]  
species6 = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31']  
#channels6 =['FSC-H','SSC-H','FL2-A','FSC-A','SSC-A','FL1-A','FL3-A']  
channels6 =["FSC-H","SSC-H","FITC-H","FSC-A","SSC-A","FITC-A","Width"]  
  
dic6={'FITC-A':'FL1-A','FITC-H':'FL2-A','Width':'FL3-A'}  
  
Ref =[references1,references2,references3,references3b,references3c,references4,references5,references6]  
Pred =[predictions1,predictions2,predictions3,predictions3b,predictions3c,predictions4,predictions5,predictions6]  
Species = [species1,species2,species3,species3b,species3c,species4,species5,species6]  
channels=[channels1,channels2,channels3,channels3b,channels3c,channels4,channels5,channels6]  
dics =[dic1,dic2,dic3,dic3b,dic3c,dic4,dic5,dic6]  
p=1  
for refs, preds, sp, channel, dic in zip(Ref,Pred, Species, channels, dics):  
 p=p+1  
 q=0  
 comb = s.f.createCombination(list(range(0, len(refs))), 2)  
 #print(refs)  
 #print(preds)  
 #print(sp)  
 #print(channel)  
 #print(dic)  
  
 for acomb in comb:  
 print(q)  
 q=q+1  
 a = acomb[0]  
 b = acomb[1]  
 if q >=0:  
 df = insilico\_com([preds[a], preds[b]], 1000)  
  
 comm='C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/test.csv'  
 saveTable(df, comm)  
 output1='C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/CELLCOGNIZE/in\_silico\_prediction4/cellcognize\_res\_' + str(p) + '\_'+str(q)+'\_' + sp[a] + '\_' + sp[b]  
 print(refs[a])  
 print(refs[b])  
 #print(comm)  
 #print(channel)  
 #print(output1)  
 if refs[a][-4:]=='.fcs' :  
 f.fcstocsv(refs[a])  
 refs2a=refs[a][:-4]+'.csv'  
 else :  
 refs2a=refs[a]  
 if refs[b][-4:]=='.fcs' :  
 f.fcstocsv(refs[b])  
 refs2b = refs[b][:-4] + '.csv'  
 else:  
 refs2b=refs[b]  
  
 #ret = eng.CellcognizeCommunity(refs2a, refs2b,comm,channel,output1) # sending input to the function  
 #phenoflow([refs[a], refs[b]], preds[a], preds[b], channel, 'C:/Users/u0128864/Documents/KU\_LEUVEN/FLOWCYTOMETRY/Articles/CLEMENCE/DATA/PHENOFLOW\_DATA/in\_silico\_prediction/pheno\_res\_'+str(p)+ '\_'+str(q)+'-'+sp[a]+'\_'+sp[b]+'.csv')  
 s.predictions([refs[a], refs[b]], [sp[a], sp[b]], [comm], ['unknown'], nbC=5000, gating=None,  
 predAn='prediction', predtype='rand',  
 ratio=1 / 7.0, repeat=10, showgat=True, average=True, doubt=0.70, save='save', fc='Accuri',  
 param=['FSC-H', 'FSC-A', 'SSC-H'], channels=channel)  
 #s.predictions([refs[a], refs[b]], [sp[a], sp[b]], [preds[a], preds[b]], [sp[a], sp[b]], nbC=5000, nbC2=1000, gating=None,  
 # predAn='analysis', predtype='neur',  
 # ratio=1 / 7.0, repeat=10, showgat=False, average=True, doubt=0.7, save='save', fc='Accuri',  
 # param=['FSC-H', 'FSC-A', 'SSC-H'], channels=channel, dicChannels=dic)

for 3 sp comm