Differential targeting analysis using PANDA and LIONESS.

Author:

Marouen Ben Guebila

In many cases, PANDA-inferred regulatory networks are compared across conditions, typically cases vs controls, or treated vs nontreated. Then we can compute the differential network between the conditions and determine which genes are differentially targeted by transcription factors (TFs) and which TFs are differentially active. In the following, we will use toy data to reconstruct wild type regulatory network, then build a knockout network through synthetically knocking down a gene by setting its expression to zero. Finally, we will determine differential gene targeting and TF activity through simple difference, and generated single sample network to obtain the statisical power to perform a statistical differential analysis.

Setting the parameters

Install netZooM

You can git clone https://github.com/netZoo/netZooM.git to your local directory. Then add it in your MATLAB path.

```
addpath(genpath('path/to/netZooM'))
```

Set the parameters

First, we start by setting the path to the 1) motif prior network, 2) the gene expression data, and 3) the ppi network data. The motif prior network is typically a TF-by-gene binary matrix where 1 indicates the presence of sequence (motif) of a TF in the gene regulatory region and 0 otherwise. Gene expression data is typically a gene-by-sample matrix containing expression data. PPI network is a TF-by-TF binary matrix, where 1 indicates a physical interaction between two TFs and 0 otherwise. If two TFs are likely to binding, they are likely ot form regulatory complexes for the same genes.

```
% Set Program Parameters
exp_file   = 'tests/test_data/expression.txt';
motif_file = 'tests/test_data/motifTest.txt';
ppi_file   = 'tests/test_data/ppi.txt';
panda_out  = '';  % optional, leave empty if file output is not required
lib_path   = '../netZooM';  % path to the folder of PANDA source code
```

For now, we will set the saving path to empty because we do not want to save the results.

```
save_temp  = '';  % optional, leave empty if temp data files will not be needed afterwa
```

Set the learning rate alpha. Typically a learning rate of 0.1 is recommended for most applications. You can save the final network in a matrix format or a pairs format.

```
alpha      = 0.1;
save_pairs = 0; %saving in .pairs format
```

Reconstruction of a wild type gene regulatory network

Now, let's reconstruct the PANDA network:

Prior1: Build coexpression network

First, we read the expression data.

```
disp('Reading in expression data!');
fid = fopen(exp_file, 'r');
headings = fgetl(fid);
n = length(regexp(headings, '\t'));
frewind(fid);
Exp = textscan(fid, ['%s', repmat('%f', 1, n)], 'delimiter', '\t');
fclose(fid);
GeneNames = Exp{1};
Exp = cat(2, Exp{2:end});
[NumGenes, NumConditions] = size(Exp);
fprintf('%d genes and %d conditions!\n', NumGenes, NumConditions);
Exp = Exp';  % transpose expression matrix from gene-by-sample to sample-by-gene
```

Then, we compute a coexpression network.

```
disp('Computing coexpression network:');
GeneCoReg = Coexpression(Exp);
```

Prior2: Build TF binding motif network

Then, we read the TF-TF PPI network.

```
disp('Reading in motif data!');
[TF, gene, weight] = textread(motif_file, '%s%s%f');
TFNames = unique(TF);
NumTFs  = length(TFNames);
[~,i]   = ismember(TF, TFNames);
[~,j]   = ismember(gene, GeneNames);
RegNetNN= zeros(NumTFs, NumGenes);
RegNetNN(sub2ind([NumTFs, NumGenes], i, j)) = weight;
fprintf('%d TFs and %d edges!\n', NumTFs, length(weight));
```

Prior3: Build TF-TF PPI network

Next, we build TF-TF PPI network that allows us to find indirect regulatory agents. In fact, some TF do not bind directly to the DNA but rather to other TFs.

```
disp('Reading in ppi data!');
TFCoop = eye(NumTFs);
if(~isempty(ppi_file))
    [TF1, TF2, weight] = textread(ppi_file, '%s%s%f');
    [~,i] = ismember(TF1, TFNames);
    [~,j] = ismember(TF2, TFNames);
    TFCoop(sub2ind([NumTFs, NumTFs], i, j)) = weight;
    TFCoop(sub2ind([NumTFs, NumTFs], j, i)) = weight;
    fprintf('%d PPIs!\n', length(weight));
end
```

Normalize the networks

To avoid biases due to the sequencing plateforma and the different scales of the measurments, we need to normalize the three prior networks.

```
disp('Normalizing Networks:');
RegNet    = NormalizeNetwork(RegNetNN);
GeneCoReg = NormalizeNetwork(GeneCoReg);
TFCoop    = NormalizeNetwork(TFCoop);
```

Calling PANDA

Fianlly, we call the PANDA function to perform the iterative optimisation procedure.

```
disp('Running PANDA algorithm:');
AgNet = PANDA(RegNet, GeneCoReg, TFCoop, alpha);
```

Let's compute the targeting score for each TF, which the weighted outdegree.

```
TFtar   = sum(AgNet,2)
```

Let's compute the targeting score for eachgene, which the weighted indegree for that gene.

```
genetar = sum(AgNet,1)
```

Simulating a gene knockout

Let's set the expression of a gene to zero to simulate a knockout. We can copy the expression table to start with:

```
ExpKo     = Exp;
```

We pick gene number 30 and set its expression to 0.

```
ExpKo(:,30) = zeros(1,1,50);
```

We compute the gene coexpression matrix and normalize it. The PPI and motif priors remain the same.

```
GeneCoReg = Coexpression(ExpKo);
GeneCoReg = NormalizeNetwork(GeneCoReg);
```

Then compute the estimate the regulatory network for the knockout.

```
AgNetKo = PANDA(RegNet, GeneCoReg, TFCoop, alpha);
```

We can compute the differential network.

```
AgNetKo - AgNet
```

Alternatively, we can compute the differential targeting score of TFs.

```
TFtarKo   = sum(AgNetKo,2)
```

We can do the same for genes:

```
genetarKo = sum(AgNetKo,1)
```

We can compute the differential targeting for TFs, which is the simple difference between the wild type profile and the knockout:

```
diffTFtar   = TFtarKo   - TFtar
```

The same for genes:

```
diffGenetar = genetarKo - genetar
```

We compute the smallest differential targeting value

```
[val,indGene]=min(diffGenetar)
indGene
```

Expectedly it corresponds to gene number 30, the one we knocked out. We can also find which TF activity was affect the most by the loss of gene 30

```
TFperturb=find(RegNetNN(:,30))
[~,tfko] =sort(diffTFtar)
TFinter=intersect(TFperturb,tfko(1:89))
```

We find that of the 89 TFs that have a binding motif to gene number 30, 83 of them were among the top 89 differentially targeted TFs after knockout of gene number 30.

Further exploratory analysis

We can also simulate 1) the knockdown of a TF through manipulating the TF-TF PPI matrix or 2) we can simulate the DNA being in closed chromatin state through setting the corresponding binding in the motif matrix to zero. We can perform differential targeting analysis to determine which parts of the networks were most affected by the perturbation.

Differential targeting analysis

Differential expression analysis

First, let's start by determining if gene expression between knockout and wild type are different. We can compute P-values through permutation

```
pvaluesCorr = mattest(Exp', ExpKo', 'Permute', 10000);
```

We find that difference between the samples is not significant, at this stage we do not to compute the magnitude of the difference or the fold change because they will be nonsignificant. In reality, when a gene is knocked out, there are more effects on the expression of other genes as well, not just its expression becoming low.

Differential targeting analysis

To do the same analysis as gene expression, we need one network per sample. To do so, we use LIONESS to generate a network for each sample of gene expression. Since the toy data used for this tutorial is used for the unit tests of netZooM, we can directly use precomputed results:

```
nSamples= size(Exp,1);
START= 1;
END  = nSamples;
ascii_out  = 0;
load('test_data/panda.test.mat');
AgNet      = AgNet';
save('panda2.test.mat','AgNet');
panda_file ='panda2.test.mat'
save_dir   = 'WT_results';
exp_file   = 'test_data/expression.transposed.mat';
motif_file = 'test_data/motif.normalized.mat';
ppi_file   = 'test_data/ppi.normalized.mat';
lioness_run(exp_file, motif_file, ppi_file, panda_file, save_dir, START, END, alpha, as
```

Once the networks generated and saved, we can load them and compute the gene targeting profiles and the TF targeting profiles for each sample in the wild type.

```
cd WT_results
TFmat   = zeros(NumTFs,nSamples);
Genemat = zeros(NumGenes,nSamples);
edgeMat = zeros(NumGenes*NumTFs,nSamples);
for i=1:nSamples
    load(['lioness.' num2str(i) '.mat']);
    TFmat(:,i)=sum(PredNet,2);
    Genemat(:,i)=sum(PredNet,1);
    edgeMat = PredNet(:);
end
cd ..
```

We can do the same for the knockout samples.

```
save('expressionKo.transposed.mat', 'ExpKo', '-v7.3');
exp_file   = 'test_data/expression.transposed.mat';
save_dir   = 'KO_results';
AgNet      = AgNetKo;
save('AgNetKo.mat','AgNet');
panda_file ='AgNetKo.mat';
lioness_run(exp_file, motif_file, ppi_file, panda_file, save_dir, START, END, alpha, as
```

Now, we load the networks and compute the gene taregting profiles and the TF targeting profiles for each sample in the knockout synthetic experiment.

```
cd KO_results
TFmatKo   = zeros(NumTFs,nSamples);
GenematKo = zeros(NumGenes,nSamples);
edgeMatKo = zeros(NumTFs*NumGenes,nSamples);
for i=1:nSamples
    load(['lioness.' num2str(i) '.mat']);
    TFmatKo(:,i)=sum(PredNet,2);
    GenematKo(:,i)=sum(PredNet,1);
    edgeMatKo = PredNet(:);
end
```

Now let's compute p-values for differential gene targeting and TF targeting.

```
pvaluesCorrGeneKo = mattest(GenematKo, Genemat, 'Permute', 10000);
pvaluesCorrTFKo   = mattest(TFmatKo, TFmat, 'Permute', 10000);
```

Edge transformation in R+

To perform fold-change analysis, usually positive values are expected. We can either threshold the network and keep the largest edges. Alternatively, we can transform the edge values to make them all positive.

One such transformation has been used in Sonawane et al [1].

```
GenematKoT1 = log(exp(GenematKo) + 1);
min(min(GenematKoT1))
max(max(GenematKoT1))
```

All the edges are now positive but they are not bounded in the right and can have infinite values. Another approach is to consider a logisitic regression with parameter 0.01 to transform edge weights into quantities between 0 and 1, which can be loosely interpreted as a probability. Let's plot a logistic regression to get a feel of the transformation.

```
b = 0.01
x = -1000:1000;
y = 1./(1+exp(-b*x));
figure;
plot(x,y)
xlabel('x')
ylabel('y')
title('Logistic regression of parameter 0.01')
```

Then we can transform the edge weights using the previous function:

```
GenematKoT2 = 1./(1+exp(-b*GenematKo));
min(min(GenematKoT2))
max(max(GenematKoT2))
```

We see that all weights are bounded between 0 and 1. We can create a volcano plot between targeting values in each sample.

```
GenematT2 = 1./(1+exp(-b*Genemat));
TFmatKoT2 = 1./(1+exp(-b*TFmatKo));
TFmatT2   = 1./(1+exp(-b*TFmat));
pvaluesCorrGeneKoT2 = mattest(GenematKoT2, GenematT2, 'Permute', 10000);
pvaluesCorrTFKoT2   = mattest(TFmatKoT2, TFmatT2, 'Permute', 10000);
SigStructureGene        = mavolcanoplot(GenematKoT2, GenematT2,pvaluesCorrGeneKoT2,'log
```

The only gene that is differentially targeted is actually gene 30

```
SigStructureGene.GeneLabels
```

Or we can assess the difference between TF targeting values:

```
SigStructureTF        = mavolcanoplot(TFmatKoT2, TFmatT2,pvaluesCorrTFKoT2,'logTrans',
```

we can compute differences between network edges after transforming through logisitic regression but this is very time consuming as it will perform the analysis on 661000 edges. Remember that PANDA and LIONESS reconstruct the regulatory network as complete bipartite graph of size nTFs*nGenes.

```
%edgeMatKo = 1./(1+exp(-b*edgeMatKo));
%edgeMat = 1./(1+exp(-b*edgeMat));
%pvaluesCorredgeKo   = mattest(edgeMatKo, edgeMat, 'Permute', 10000);
```

Alternatively, you can also use limma package to perform a linear regression and account for covariates when performing the differential analysis such as sex or age.

References

[1] Sonawane, Abhijeet Rajendra, et al. "Understanding tissue-specific gene regulation." *Cell reports* 21.4 (2017): 1077-1088