TallerOMICASfinal

December 12, 2019

1 Coexpression networks in the identification of genes that respond to saline stress

Author

Camila Riccio Rengifo Pontificia Universidad Javeriana, Cali Optimización Multiescala Insilico de Cultivos Agrícolas Sostenibles (ÓMICAS), P5 camila.riccio@javerianacali.edu.co

2 Introduction

A gene co-expression network is an undirected graph , where each node correspond to a gene, and a pair of nodes is connected if there is a significant co-expression relationship between them, that is, if they show a similar expression pattern through all samples. These co-expression networks are of biological interest since the co-expressed genes are usually controlled by the same transcriptional regulatory pathway, are functionally related or are members of the same pathway or metabolic complex.

The co-expression network is constructed from the expression levels of the genes under a specific condition or on their change of expression between two different conditions (i.e. control and stress).

To study the response to saline stress in rice from a co-expression network, a relationship is established with the levels of Na / K in the samples as an indicator of salinity tolerance, which allows identifying the most significant genes in the process.

2.1 objectives:

- Integrate RNA-seq data under control and saline stress into a co-expression network.
- Detect gene modules with similar expression change patterns (LogFoldChange).
- Match the modules with a relevant phenotypic characteristic in the response to saline stress (Na/K level in the plant) and select the most relevant ones.

3 Import libraries

[0]:
```
import pandas as pd
import sys
import numpy as np
import warnings
```

```
warnings.filterwarnings("ignore")
```

[0]:
```
%load_ext rpy2.ipython
```

```
The rpy2.ipython extension is already loaded. To reload it, use:
  %reload_ext rpy2.ipython
```

[0]:
```
%%R
# install.packages("BiocManager")
# BiocManager::install("WGCNA")
install.packages("WGCNA")
library(WGCNA)
```

[0]:
```
%%R
install.packages("caret")
install.packages("glmnet")
```

[0]:
```
%%R
# Loading required R packages
library(tidyverse)  #for easy data manipulation and visualization
library(caret)      #for easy machine learning workflow
library(glmnet)     #for computing penalized regression
```

4 Prepare data from RNA-seq

RNA-seq data was accessed through GEO database [**?**] (Accession number GSE98455), corresponding to $n = 57845$ gene expression profiles of shoot tissues measured for both control and salt condition in $p = 92$ diverse rice accessions of the Rice Diversity Panel 1. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE98455

4.1 Load Expression file

[0]:
```
# Load complete expression file
df_all = pd.read_csv('RNASeqData.txt',  '\t', index_col=0)
df_all = df_all.iloc[:,:df_all.shape[1]-1]
print(df_all.shape)
df_all.head()
```

```
(57845, 368)
```

[0]:

Gene	GSM2596381_101_C_rep1	...	GSM2596760_9_S_rep2
		...	
13103.t02982	0.0	...	0.0
13105.t01662	1.0	...	0.0
13110.t02303	261.0	...	144.0
13108.t00264	25.0	...	23.0
13102.t01556	80.0	...	59.0

```
[5 rows x 368 columns]
```

[0]:
```
# Randomly select only 10000 genes
df = df_all.sample(10000)
print(df.shape)
df.head()
```

```
(10000, 368)
```

[0]:
```
             GSM2596381_101_C_rep1  ...  GSM2596760_9_S_rep2
Gene                                ...
13108.t04075                 138.0  ...                132.0
13104.t04582                   3.0  ...                  8.0
13108.t03131                  68.0  ...                 84.0
13110.t02328                   0.0  ...                  0.0
13101.t03278                   0.0  ...                  0.0

[5 rows x 368 columns]
```

4.2 DESeq normalization

[0]:
```
def DESeq2(df):
    '''df: dataframe with expression level of genes'''
    # step 1: take log of all values
    df_deseq = df.apply(np.log)
    # step 2: Average each raw
    geometric_average = df_deseq.mean(axis=1)
    # Step 3: Filter out genes with Infinity
    df_deseq = df_deseq[geometric_average!=-np.inf]
    # Step 4: Subtract the average log value from the log(count)
    df_deseq = df_deseq.sub(df_deseq.mean(axis=1), axis=0)
    # Step 5: Calculate the median of the ratios for each sample (column)
    medians = df_deseq.median(axis=0)
    # Step 6: Convert the medians to "normal numbers" to get the final
    # scaling factors for each sample
    scaling_factors = np.exp(medians)
    # Divide the original read counts by the scaling factors
    df_deseq = df.div(scaling_factors, axis=1)
    return df_deseq
```

[0]:
```
df = DESeq2(df)
df.head()
```

[0]:
```
             GSM2596381_101_C_rep1  ...  GSM2596760_9_S_rep2
Gene                                ...
13108.t04075             162.709699  ...           129.429271
13104.t04582               3.537167  ...             7.844198
```

```
13108.t03131              80.175794  ...            82.364081
13110.t02328               0.000000  ...             0.000000
13101.t03278               0.000000  ...             0.000000

[5 rows x 368 columns]
```

4.3 Average repetitions from each accession

[0]:
```
cols = ['_'.join(c.split('_')[:2]) for c in df.columns.tolist()]
num_rep = 2
df_av = pd.DataFrame()
# every 4 columns there is a different accession
# every 2 columns there is a different condition (control <-> stress)
for i in range(0,df.shape[1]-3,num_rep*2):
    df_av[cols[i]]=(df.iloc[:,i].values + df.iloc[:,i+1])/2
    df_av[cols[i+2]]=(df.iloc[:,i+2].values + df.iloc[:,i+3])/2

df_av.head()
```

[0]:
```
             GSM2596381_101  GSM2596383_101  ...  GSM2596757_9  GSM2596759_9
Gene                                         ...
13108.t04075      175.077425      145.193552  ...    159.215364    135.059188
13104.t04582        9.441777        5.965586  ...      3.398467      6.464673
13108.t03131       76.809608       68.419022  ...     82.089730     65.760258
13110.t02328        0.000000        0.000000  ...      0.000000      0.000000
13101.t03278        0.000000        0.000000  ...      0.000000      0.000000

[5 rows x 184 columns]
```

4.4 Remove genes with low expression

for more than 80% samples, normalized read count smaller than 10

[0]:
```
print(df_av.shape)
q = np.array(df_av.quantile(0.8,axis = 1))
df_av = df_av[q>=10]
print(df_av.shape)
```

```
(10000, 184)
(3947, 184)
```

4.5 Remove genes with low variance:

The ratio of upper quantile to lower quantile of normalized read count smaller than 1.5

[0]:
```
uq = df_av.quantile(0.75,axis = 1)
lq = df_av.quantile(0.25,axis = 1)
ratio = np.array([(u+1)/(l+1) for u,l in zip(uq,lq)])
```

```
df_av = df_av[ratio>1.5]
print(df_av.shape)
```

```
(1639, 184)
```

4.6 Separate Control and Stress data

[0]:
```
cols = df_av.columns.tolist()
control,stress = pd.DataFrame(), pd.DataFrame()
for i in range(0,df_av.shape[1],2):
    control[cols[i]]=df_av.iloc[:,i]
    stress[cols[i+1]]=df_av.iloc[:,i+1]

control.head()
```

[0]:
```
              GSM2596381_101  GSM2596385_105  ...  GSM2596753_91  GSM2596757_9
Gene                                          ...
13102.t00559        7.290879       12.067669  ...      10.590461      6.499051
13111.t00059      213.010304      111.580497  ...     104.086882    130.973646
13112.t00015       19.339423       23.267933  ...      27.333780      0.000000
13104.t04551       43.054198       85.629922  ...      34.493637     45.256822
13101.t05507      661.468112      690.949902  ...     623.163905   1047.488158

[5 rows x 92 columns]
```

[0]:
```
stress.head()
```

[0]:
```
              GSM2596383_101  GSM2596387_105  ...  GSM2596755_91  GSM2596759_9
Gene                                          ...
13102.t00559        7.577417        9.927047  ...      11.531403      9.273248
13111.t00059      200.254736       66.429743  ...      15.716341     75.879628
13112.t00015       18.035716       16.594511  ...      41.620217      0.000000
13104.t04551       54.987205       87.205817  ...      41.334216     51.375697
13101.t05507      669.897218      980.101515  ...     872.577327   1076.465750

[5 rows x 92 columns]
```

4.7 Log Fold Change

The Fold change is a measure describing how much a quantity changes going from an initial to a final value (divide the salt count with corresponding control count).

If you use log-transformed expression values, you model PROPORTIONAL changes rather than additive changes. This is typically biologically more relevant.

A doubling (or the reduction to 50%) is often considered as a biologically relevant change. On the log2 scale this translates to one unit (+1 or -1)

[0]:
```
colnames = [c.split('_')[0] for c in control.columns.tolist()]

Log2FC = pd.DataFrame()
```

```
for i in range(0,control.shape[1]):
    Log2FC[colnames[i]] = [np.log2((s+1)/(c+1)) for s,c in zip(stress.iloc[:
 ↪,i],control.iloc[:,i])]

print(Log2FC.shape)

Log2FC.index = control.index.tolist()
Log2FC.head()
```

```
(1639, 92)
```

[0]:

	GSM2596381	GSM2596385	...	GSM2596753	GSM2596757
13102.t00559	0.049018	-0.258098	...	0.112610	0.454112
13111.t00059	-0.088658	-0.739500	...	-2.652252	-0.779577
13112.t00015	-0.095570	-0.463926	...	0.589015	0.000000
13104.t04551	0.345818	0.026008	...	0.254264	0.179231
13101.t05507	0.018241	0.503735	...	0.485010	0.039331

[5 rows x 92 columns]

4.8 Remove genes exhibiting low Log2Fold change variance

For this log2 fold change matrix used for coexpression network construction, genes with the ratio of upper quantile to lower quantile larger than 0.25 were kept.

[0]:
```
uq = Log2FC.quantile(0.75,axis = 1) #upper quantil
lq = Log2FC.quantile(0.25,axis = 1) #lower quantil

ratio = np.array([u-l for u,l in zip(uq,lq)])
Log2FC = Log2FC[ratio>0.25]
print(Log2FC.shape)
Log2FC.head()
```

```
(1565, 92)
```

[0]:

	GSM2596381	GSM2596385	...	GSM2596753	GSM2596757
13102.t00559	0.049018	-0.258098	...	0.112610	0.454112
13111.t00059	-0.088658	-0.739500	...	-2.652252	-0.779577
13112.t00015	-0.095570	-0.463926	...	0.589015	0.000000
13104.t04551	0.345818	0.026008	...	0.254264	0.179231
13101.t05507	0.018241	0.503735	...	0.485010	0.039331

[5 rows x 92 columns]

[0]:
```
Log2FC = Log2FC.transpose()
```

[0]:
```
Log2FC.shape
```

[0]:
```
(92, 1565)
```