ómicas

Hace constar que:

Jorge Finke

Participó en el 1er Taller Anual: Del Gen al Cultivo como PONENTE – con la presentación de: Aprendizaje automático en bioinformática

En constancia de lo anterior, se firma en Santiago de Cali a los siete (07) días del mes de diciembre de 2019.

[illegible] [illegible]

Andrés Jaramillo Botero
Director Científico

Luis Eduardo Tobón
Subdirector Fortalecimiento Institucional

Apoyan:

El futuro es de todos | Gobierno de Colombia

COLOMBIA CIENTÍFICA
Conocimiento Global para el Desarrollo

Taller ÓMICAS 2019 - Del gen al cultivo

Taller 5: Aprendizaje Automático en Bioinformática

1. Definiciones:

Gen: Unidad de información en un locus de ácido desoxirribonucleico (ADN) que codifica un producto génico, ya sea proteínas o ARN. Es la unidad molecular de la herencia genética, pues almacena la información genética y permite transmitirla a la descendencia. Los genes se encuentran en los cromosomas, y cada uno ocupa en ellos una posición determinada llamada locus.

Expresión Genética: Proceso por medio del cual todos los microorganismos procariotas y células eucariotas transforman la información codificada por los ácidos nucleicos en las proteínas necesarias para su desarrollo, funcionamiento y reproducción con otros organismos. La expresión génica es clave para la creación de un fenotipo.

Red de Coexpresión de Genes: Grafo no dirigido, cuyos *nodos* representan genes y las conexiones entre dos nodos, conocidas como *arcos*, representan una relación significativa de coexpresión entre un par de genes.

2. Importación de datos

Se utilizará un subconjunto sintético de 50 genes del Arroz, cada uno con 2678 datos de expresión.

Para ello, por favor cargar el archivo "dataset.csv" en su propio Drive de Google. Luego de esto, acceder a Google Drive desde *Colaboratory*

In [0]:
```
"""
ÓMICAS hereby disclaims all copyright interest in this code written by Nicolás Lópe
z.
"""

from google.colab import drive
from math import *
import time
from matplotlib import pyplot as plt
import numpy as np
from tqdm import tqdm
from scipy.spatial.distance import pdist, squareform

drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_i
d=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redir
ect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20htt
ps%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.
com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.read
only%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
```

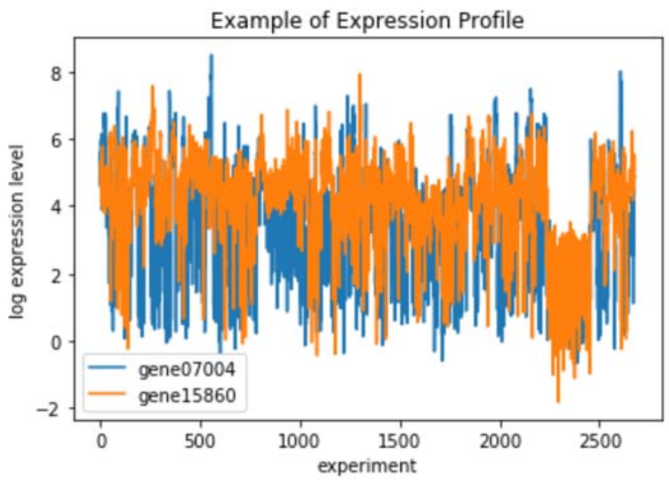
Ahora, se cargan en Python los datos de expresión

In [0]:
```
names, expression = [], []
with open('/content/drive/My Drive/dataset.csv', 'r') as f:
  # reading expression data
  for line in f.readlines():
    name, tmp_ = line.strip().split(',', 1)
    vals = [log(float(x)) for x in tmp_.split(',')]
    names.append(name)
    expression.append(vals)

print('data succesfully loaded')
```

```
data succesfully loaded
```

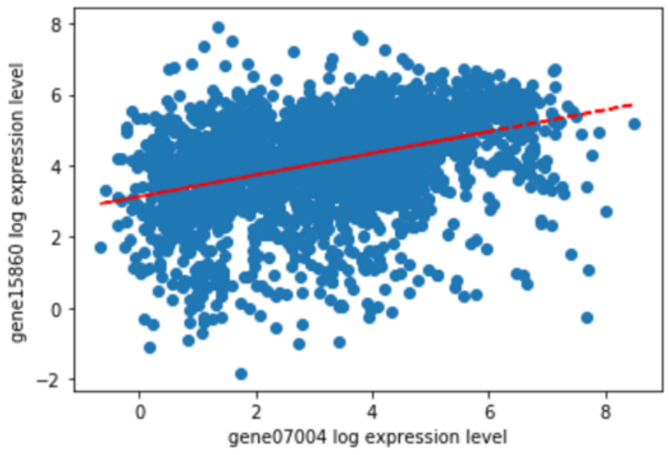
Vamos a visualizar las dos primeras series de datos de expresión mediante la biblioteca ***pyplot***, de ***matplotlib*** (se usa el alias *plt* para simplicidad en el código)

In [0]:
```
plt.plot(expression[0], label=names[0])
plt.plot(expression[1], label=names[1])
plt.title('Example of Expression Profile')
plt.ylabel('log expression level')
plt.xlabel('experiment')
plt.legend()
plt.show()
```



Ahora, vamos a visualizar esos mismos 2 genes pero como una gráfica de dispersión utilizando *plt.scatter*. Vamos a agregar una linea de tendencia utilizando la biblioteca ***numpy***

```
In [0]: # scatter plot
        plt.scatter(expression[0], expression[1])
        plt.xlabel(names[0] + ' log expression level')
        plt.ylabel(names[1] + ' log expression level')

        # linear fit visualization
        fit = np.polyfit(expression[0], expression[1], 1)
        polynomial = np.poly1d(fit)
        plt.plot(expression[0], polynomial(expression[0]), "r--")

        # show full plot
        plt.show()
```



3. Métricas de Dependencia Lineal entre Variables

3.1. Coeficiente de Correlación de Pearson (PCC)

$$\rho_{X,Y} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

Donde:

σ_{XY} es la covarianza de (X, Y)

σ_X es la desviación estándar de la variable X

σ_Y es la desviación estándar de la variable Y

De manera análoga podemos calcular este coeficiente sobre un estadístico muestral, denotado como r_{xy} así:

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\ \sqrt{n\sum y_i^2 - (\sum y_i)^2}}.$$

Interpretación

PCC puede tomar valores entre $[-1, 1]$, y su interpretación puede descomponerse dos partes:

- *Signo*: El signo de PCC indica si la correlación es creciente $(PCC > 0)$ o decreciente $(PCC < 0)$. Si $(PCC > 0)$, entonces a medida que X aumenta, Y también lo hace.
- *Magnitud*: La magnitud de PCC indica la fuerza de la correlación. Si $|PCC| = 0$, entonces se dice que las variables X e Y no presentan dependencia lineal. Por otra parte, si $|PCC| = 1$, entonces se dice que X e Y tienen dependencia lineal perfecta.

Para más información: https://es.wikipedia.org/wiki/Coeficiente_de_correlación_de_Pearson (https://es.wikipedia.org/wiki/Coeficiente_de_correlación_de_Pearson)

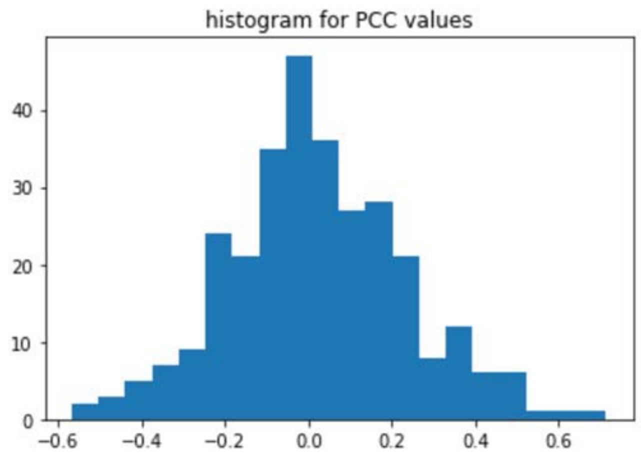
In [0]:
```
def PCC(X, Y):
  return np.corrcoef(X, Y)[0, 1]

a = time.time()
print("PCC between {} and {} is {:.8f}".format(names[0], names[1],
                                              PCC(expression[0], expression[1])))
print("Elapsed time:", time.time()-a)
```

```
PCC between gene07004 and gene15860 is 0.37932066
Elapsed time: 0.0024025440216064453
```

Ahora, vamos a realizar el cálculo de todos los valores de coexpresión utilizando PCC, para luego realizar un histograma de la distribución de PCC. Vamos a importar la biblioteca ***tqdm*** de *Python* para poder visualizar el progreso de las operaciones

In [0]:
```python
PCC_results = []
PCC_graph = []
for i in tqdm(range(len(names)-1), desc="approx. progress"):
  for j in range(i + 1, len(names)):
    value = PCC(expression[i], expression[j])
    PCC_results.append(value)
    PCC_graph.append((value, i, j))

plt.hist(PCC_results, bins=20)
plt.title("histogram for PCC values")
plt.show()
```

```
approx. progress: 100%|██████████| 24/24 [00:00<00:00, 198.98it/s]
```



3.2 Biweight Midcorrelation (BiCor)

Esta métrica es una medida de similitud entre muestras. Se basa en la mediana y no en la media, haciéndola menos sensible a *outliers*. Puede ser una alternativa robusta a otras métricas de similitud (PCC, MI, ...)

$$u_i = \frac{x_i - \mathrm{med}(x)}{9\,\mathrm{mad}(x)},$$
$$v_i = \frac{y_i - \mathrm{med}(y)}{9\,\mathrm{mad}(y)}$$

- $med(\cdot)$ es la mediana y $mad(\cdot)$ es la desviación absoluta de la mediana, y se calcula así: $mad(x) = med(|x_i - med(x)|)$

$$w_i^{(x)} = \left(1 - u_i^2\right)^2 I\left(1 - |u_i|\right)$$
$$w_i^{(y)} = \left(1 - v_i^2\right)^2 I\left(1 - |v_i|\right)$$

- donde

$$I(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

Luego, se normaliza para que los pesos sumen 1:

$$\tilde{x}_i = \frac{\left(x_i - \mathrm{med}(x)\right) w_i^{(x)}}{\sqrt{\sum_{j=1}^{m}\left[(x_j - \mathrm{med}(x)) w_j^{(x)}\right]^2}}$$
$$\tilde{y}_i = \frac{\left(y_i - \mathrm{med}(y)\right) w_i^{(y)}}{\sqrt{\sum_{j=1}^{m}\left[(y_j - \mathrm{med}(y)) w_j^{(y)}\right]^2}}.$$

Por último, se calcula la métrica como: $\mathrm{bicor}\left(x, y\right) = \sum_{i=1}^{m} \tilde{x}_i \tilde{y}_i$

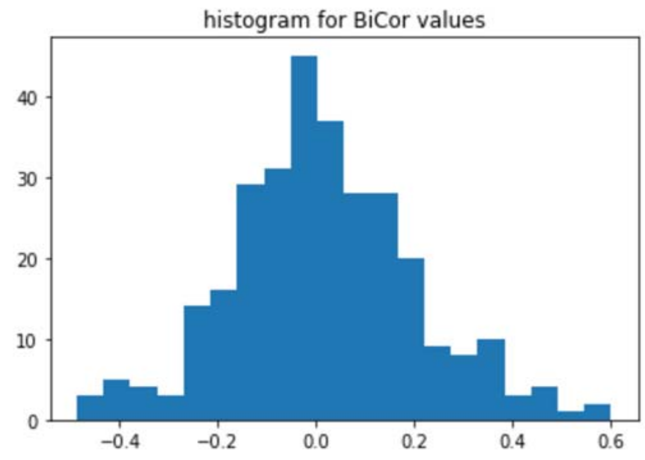
In [0]:
```
def BiCor(X, Y):
  # calculating median
  n = len(X)
  medx = np.median(X)
  medy = np.median(Y)

  #calculating MAD
  tmp = [abs(x - medx) for x in X]
  madx = np.median(tmp)

  tmp = [abs(y - medy) for y in Y]
  mady = np.median(tmp)

  # calculating U and V
  U = [(x - medx)/(9.0 * madx) for x in X]
  V = [(y - medy)/(9.0 * mady) for y in Y]
  Wx = [((1 - u**2)**2 if 1 - abs(u) > 0.0 else 0.0) for u in U]
  Wy = [((1 - v**2)**2 if 1 - abs(v) > 0.0 else 0.0) for v in V]

  #now, calculating X overline and Y overline
  denom_x = sqrt(sum([Wx[i]*(X[i] - medx)**2 for i in range(n)]))
  denom_y = sqrt(sum([Wy[i]*(Y[i] - medy)**2 for i in range(n)]))
  Xo = [(X[i] - medx) * Wx[i] / denom_x for i in range(n)]
  Yo = [(Y[i] - medy) * Wy[i] / denom_y for i in range(n)]
  ans = sum([Xo[i] * Yo[i] for i in range(n)])
  return ans

a = time.time()
print("BiCor between {} and {} is {:.8f}".format(names[0], names[1],
                                          BiCor(expression[0], expression[1])))
print("Elapsed time:", time.time()-a)
```

```
BiCor between gene07004 and gene15860 is 0.34035646
Elapsed time: 0.03493809700012207
```

Ahora, se calcula para ***BiCor*** para todas las posibles relaciones de coexpresión en nuestro conjunto de datos:

```
In [0]: BiCor_results = []
        BiCor_graph = []
        for i in tqdm(range(len(names)-1), desc="approx. progress", ascii=True):
          for j in range(i + 1, len(names)):
            value = BiCor(expression[i], expression[j])
            BiCor_results.append(value)
            BiCor_graph.append((value, i, j))

        plt.hist(BiCor_results, bins=20)
        plt.title("histogram for BiCor values")
        plt.show()
```

```
approx. progress: 100%|##########| 24/24 [00:06<00:00,  3.99it/s]
```



4. Métricas de Dependencia No Lineal entre Variables

4.1 Distance Correlation (dCor)

La correlación de la distancia es una medida de dependencia entre dos variables entre dos vectores aleatorios. Permite encontrar relaciones lineales o no lineales. Se basa en 2 métricas llamadas varianza de la distancia $(dVar)$ y covarianza de la distancia $(dCov)$.

$$\operatorname{dCov}^2(X,Y) := \frac{1}{n^2}\sum_{j=1}^{n}\sum_{k=1}^{n} A_{j,k}\,B_{j,k}.$$

$$\operatorname{dVar}^2_n(X) := \operatorname{dCov}^2_n(X,X) = \tfrac{1}{n^2}\sum_{k,\ell} A^2_{k,\ell},$$

$$\operatorname{dCor}(X,Y) = \frac{\operatorname{dCov}(X,Y)}{\sqrt{\operatorname{dVar}(X)\,\operatorname{dVar}(Y)}},$$

donde $A_{j,k}$, $B_{j,k}$ corresponden a matrices de distancia doblemente centradas para las variables X e Y respectivamente.

Para más información: https://en.wikipedia.org/wiki/Distance_correlation#Distance_correlation (https://en.wikipedia.org/wiki/Distance_correlation#Distance_correlation)

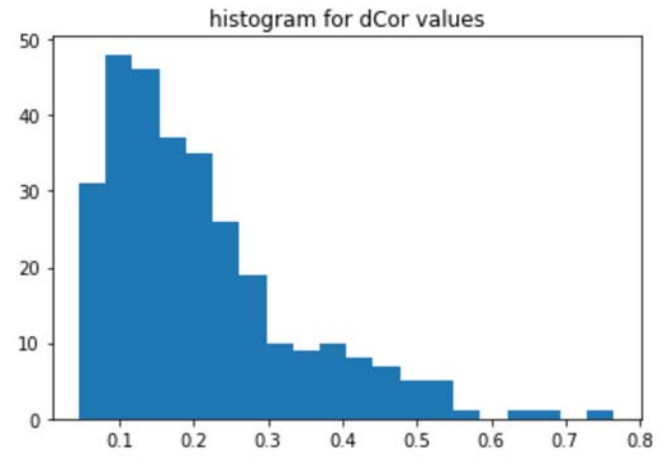
In [0]:
```python
def dCor(X, Y):
  # preparating data
  X = np.atleast_2d([[x] for x in X])
  Y = np.atleast_2d([[y] for y in Y])
  n = len(X)

  # calculating A and B matrices
  a = squareform(pdist(X))
  b = squareform(pdist(Y))
  A = a - a.mean(axis=0)[None, :] - a.mean(axis=1)[:, None] + a.mean()
  B = b - b.mean(axis=0)[None, :] - b.mean(axis=1)[:, None] + b.mean()

  # calculating metrics
  dcov2_xy = (A * B).sum()/float(n * n)
  dcov2_xx = (A * A).sum()/float(n * n)
  dcov2_yy = (B * B).sum()/float(n * n)
  dcor = np.sqrt(dcov2_xy)/np.sqrt(np.sqrt(dcov2_xx) * np.sqrt(dcov2_yy))
  return dcor

a = time.time()
print("dCor between {} and {} is {:.8f}".format(names[0], names[1],
                                                 dCor(expression[0], expression[1])))
print("Elapsed time:", time.time()-a)
```

```
dCor between gene07004 and gene15860 is 0.38458348
Elapsed time: 0.42993855476379395
```

Ahora, se calculan todos los valores de coexpresión con ***dCor***, de manera similar que con ***PCC***

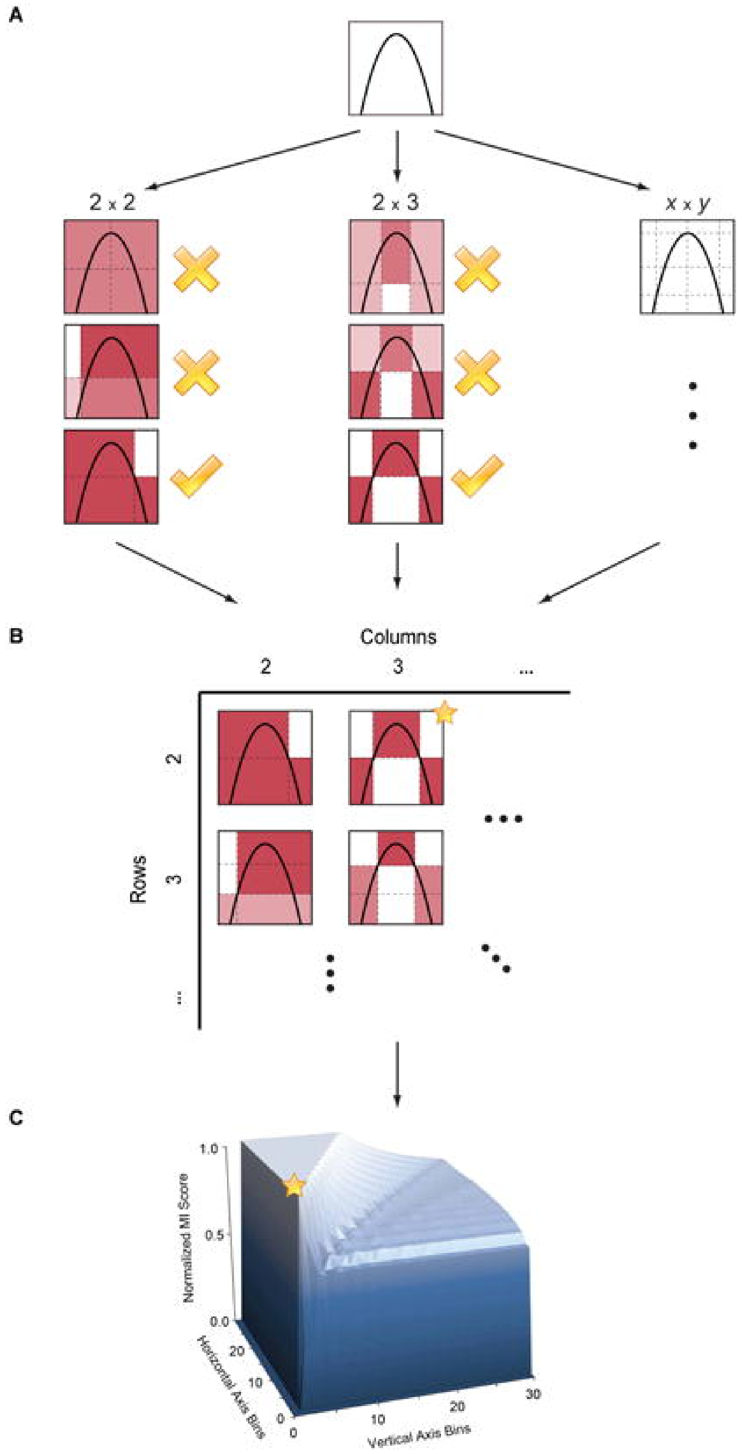
In [0]:
```python
dCor_results = []
dCor_graph = []
for i in tqdm(range(len(names)-1), desc="approx. progress", ascii=True):
  for j in range(i + 1, len(names)):
    value = dCor(expression[i], expression[j])
    dCor_results.append(value)
    dCor_graph.append((value, i, j))

plt.hist(dCor_results, bins=20)
plt.title("histogram for dCor values")
plt.show()
```

```
approx. progress: 100%|##########| 24/24 [02:09<00:00,  1.42s/it]
```



4.2. Maximal Information Coefficient (MIC)

El *MIC* es una medida de la fuerza de la asociación lineal o no lineal entre dos variables X e Y. Se basa en la Teoría de la Información, y busca maximizar la Información Mutua de dos variables aleatorias continuas mediante un esquema de *binning*, escogiendo la cantidad adecuada de "cajas" en cada una de las variables.



Para más información: [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3325791/ (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3325791/)](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3325791/),

Primero, se debe instalar el paquete ***minepy***, mediante el comando:

```
In [0]: !pip install minepy
```

```
Collecting minepy
  Downloading https://files.pythonhosted.org/packages/90/a6/cdfe0f50b16d18196b3a
21d3df8c06c361ccbd553717093133a88227823b/minepy-1.2.4.tar.gz (493kB)
     |████████████████████████████████| 501kB 4.8MB/s
Requirement already satisfied: numpy>=1.3.0 in /usr/local/lib/python3.6/dist-pac
kages (from minepy) (1.17.4)
Building wheels for collected packages: minepy
  Building wheel for minepy (setup.py) ... done
  Created wheel for minepy: filename=minepy-1.2.4-cp36-cp36m-linux_x86_64.whl si
ze=174016 sha256=d7c4bf273695889d85ae28ca7fc65cf197c4bb8815ad61fecff2e5720049cfb
3
  Stored in directory: /root/.cache/pip/wheels/ea/ad/3a/0e6f5c87be5ee6ad987bd7a3
17dd6b92e616d559f63f4d8acc
Successfully built minepy
Installing collected packages: minepy
Successfully installed minepy-1.2.4
```

```
In [0]: from minepy import MINE

mine = MINE(alpha=0.6, c=15, est="mic_approx")

def MIC(X, Y):
  mine.compute_score(X, Y)
  return mine.mic()

a = time.time()
print("MIC between {} and {} is {:.8f}".format(names[0], names[1],
                                        MIC(expression[0], expression[1])))
print("MIC between {} and {} is {:.8f}".format(names[0], names[2],
                                        MIC(expression[0], expression[2])))
print("MIC between {} and {} is {:.8f}".format(names[1], names[2],
                                        MIC(expression[1], expression[2])))
print("Elapsed time:", time.time()-a)
```

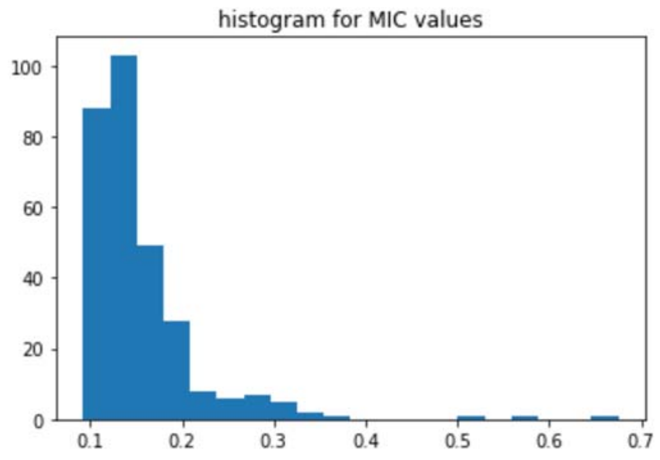
```
MIC between gene07004 and gene15860 is 0.18729382
MIC between gene07004 and gene05811 is 0.15608590
MIC between gene15860 and gene05811 is 0.19939324
Elapsed time: 2.8735692501068115
```

Ahora, se calcula toda la red como en los casos anteriores. Para efectos de eficiencia, en este caso se utilizará la función *pstats*:

In [0]:

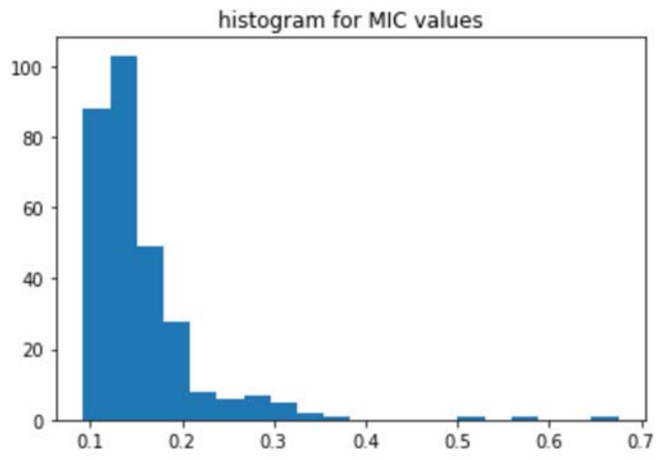
```
import minepy

MIC_results = []
MIC_graph = []

# calculating all coexpression values at once
tic = time.time()
tmp, _ = minepy.pstats(expression, alpha=0.6, c=15, est="mic_approx")
toc = time.time()
print("Elapsed time:", toc-tic)
print(tmp[0], tmp[1], tmp[23], tmp[24], tmp[25])
# labeling results
k = 0
for i in range(len(names) - 1):
  for j in range(i + 1, len(names)):
    value = tmp[k]
    MIC_results.append(value)
    MIC_graph.append((value, i, j))
    k += 1

plt.hist(MIC_results, bins=20)
plt.title("histogram for MIC values")
plt.show()
```

```
Elapsed time: 295.80040979385376
0.18729382166119615 0.15608589775414297 0.14411637268049754 0.19939324255346283
0.20481740421286743
```



In [0]:
```python
import minepy

MIC_results = []
MIC_graph = []

# calculating all coexpression values at once
tic = time.time()
for i in tqdm(range(len(names) - 1), desc="MIC", ascii=True):
  for j in range(i + 1, len(names)):
    value = MIC(expression[i], expression[j])
    if j<=2: print(i, j, value)
    MIC_results.append(value)
    MIC_graph.append((value, i, j))

plt.hist(MIC_results, bins=20)
plt.title("histogram for MIC values")
plt.show()
```

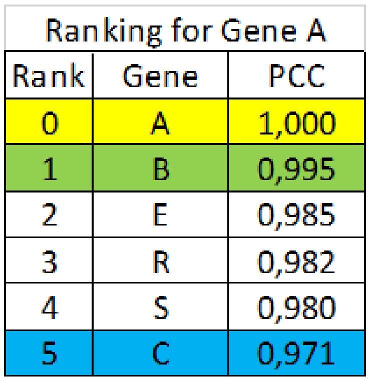
```
MIC:   0%|          | 0/24 [00:00<?, ?it/s]

0 1 0.18729382166119615
0 2 0.15608589775414297

MIC:   4%|4         | 1/24 [00:23<08:59, 23.47s/it]

1 2 0.19939324255346283

MIC: 100%|##########| 24/24 [04:54<00:00,  3.25s/it]
```



5. Jerarquización Mutua (Mutual Rank, MR)

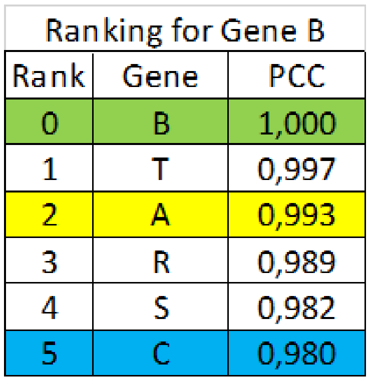
Obayashi y Kinoshita(2009, 2010) muestran que es mejor utilizar el ranking de los valores de correlación que utilizar los valores de correlación en las redes de coexpresión.

De esta forma, el valor de coexpresión de una pareja de genes no es tan importante como la posición relativa de este valor respecto a los demás. Por lo anterior, es posible comparar métricas cuyos valores tengan rangos diferentes.



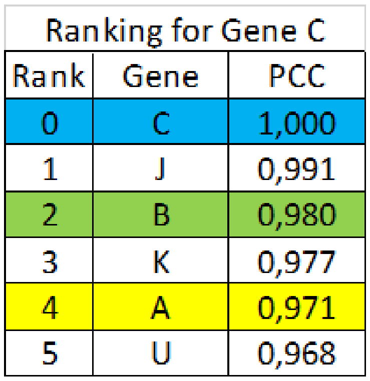
Ranking for Gene A

Rank	Gene	PCC
0	A	1,000
1	B	0,995
2	E	0,985
3	R	0,982
4	S	0,980
5	C	0,971



Ranking for Gene B

Rank	Gene	PCC
0	B	1,000
1	T	0,997
2	A	0,993
3	R	0,989
4	S	0,982
5	C	0,980



Ranking for Gene C

Rank	Gene	PCC
0	C	1,000
1	J	0,991
2	B	0,980
3	K	0,977
4	A	0,971
5	U	0,968

$R_{A \to A} = R_{B \to B} = R_{C \to C} = 0$

$R_{A \to B} = 1$, $R_{B \to A} = 2$

$\to MR_{A,B} = \sqrt{R_{A \to B} \cdot R_{B \to A}} = 1.414$

ÓMICAS

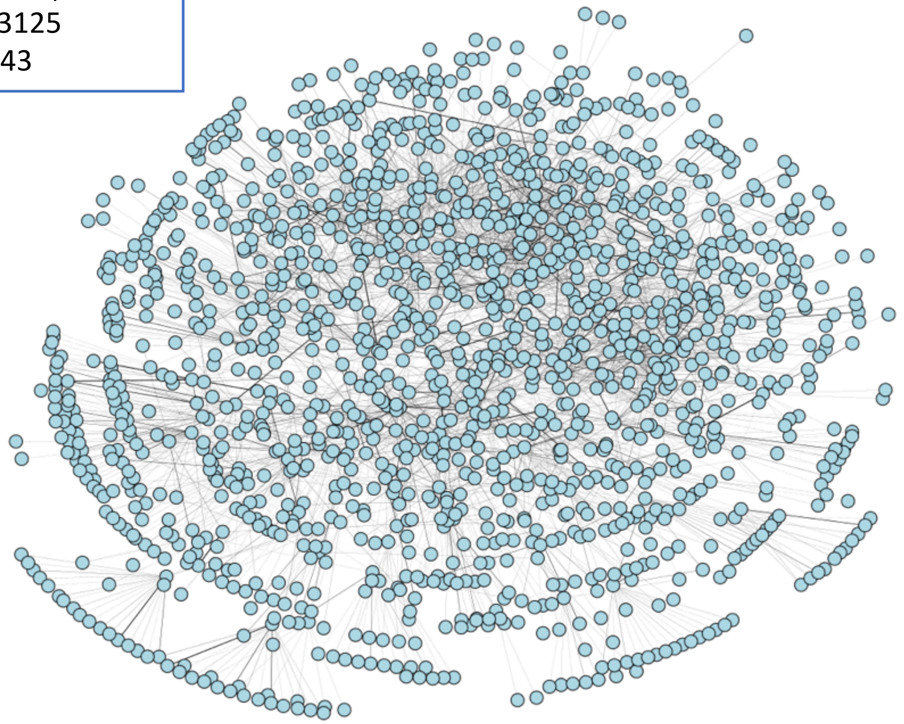
Camilo Rocha (co-IP)
Jorge Finke (co-IP)
Camila Riccio (doctorando)
Miguel Romero (doctorando)
Nicolás López (doctorando)

www.omicas.co

Mejoramiento in-silico de cultivos a partir de la caracterización ómica multi-escala

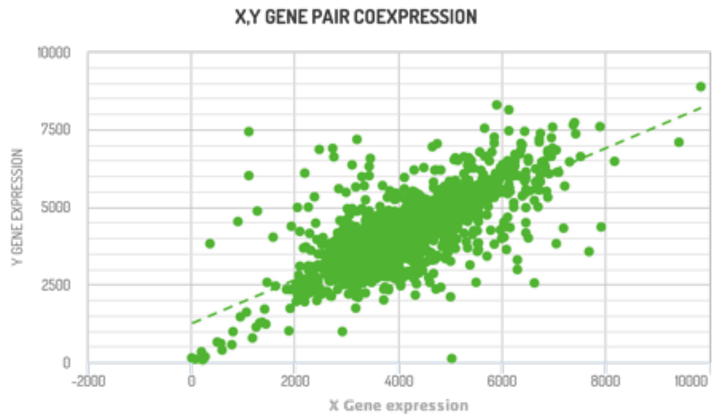
Resultados – annotación de genes

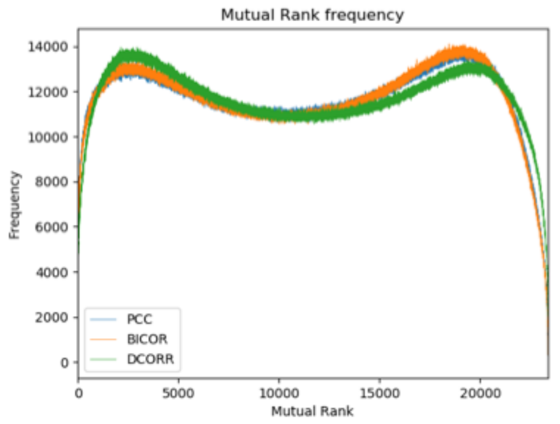
Red de co-expresión (Pearson)
Número de genes (nodos): 19795
Número de arcos: 553125
Genes anotados: 10743



ID	Biological process	# Genes	Max FP	# FP
0006807	Nitrogen compound metabolic process	15	41	1
0006289	Nucleotide-excision repair	20	46	1
0006397	mRNA processing	17	48	1
0007017	Microtubule-based process	18	49	1

Métrica	Número de nodos	Número de arcos	Promedio vecinos	Componentes	Diámetro
PCC	19924	90281	9.06	197	18
BICOR	19834	87831	8.86	220	20
DCORR	16400	57487	7.01	32	16





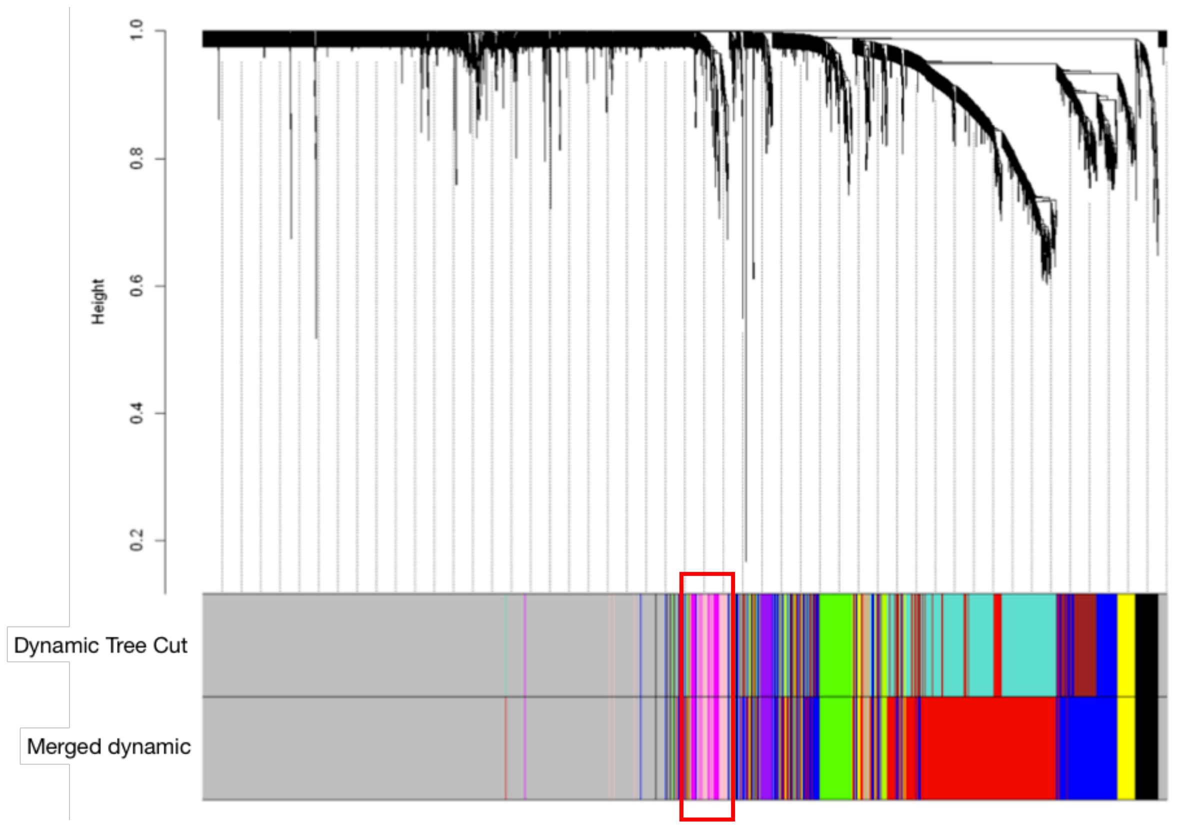
In-silico

El futuro es de todos
Gobierno de Colombia
COLOMBIA CIENTÍFICA
ómicas

Mejoramiento in-silico de cultivos a partir de la caracterización ómica multi-escala

Resultados – respuesta a estrés

Respuesta a estrés salino:
Genes: 71



Miguel Romero

Aplicación: Identificación de funciones biológicas a través de características topológicas de redes de co-expresión génica

- Balance de clases de datos
- Validación cruzada
- Árboles impulsados por gradiente

Nicolás López

Aplicación: Generación de redes de co-expresión basadas en métricas alternativas

- Evaluación de métricas alternativas (Pearson)
- *Bi-weighted correlation*
- *Distance correlation*
- *Mutual information content*
- *Mutual Rank*

Camila Riccio

Aplicación: Identificación de rasgos fenotípicos

- Respuesta al estrés salino
- Técnicas de mínimo cuadrado ordinario (LASSO)

Gene Functional Annotation Prediction

Author

Miguel Ángel Romero
Pontificia Universidad Javeriana, Cali
Optimización Multiescala In-silico de Cultivos Agrícolas Sostenibles (ÓMICAS), P5
miguel.romero@javerianacali.edu.co

Introduction

In this kernel we will use supervised machine learning models to see how accurate they are in predicting functional gene annotations. The dataset includes existing knowledge body of gene annotations of the Oryza Sativa Japonica (a variety of rice) genome and the topological properties of its gene co-expression network. The supervised machine learning models are designed to discover unknown annotations. Let's start!

Our Goals:

- Understand the data.
- Create a 50/50 sub-dataframe ratio of "Annotated" and "Non-Annotated" genes.
- Determine the sampling method we are going to use and decide which one has a higher accuracy.
- Understand the importance of the topological properties.
- Determine promising candidates to carry out further studies through in-vivo experiments.

Outline:

I. **Understanding our data**
a) Gather Sense of our Data

II. **Preprocessing**
a) Scaling
b) Correlation
c) Splitting the Data

III. **Random UnderSampling and Oversampling**
a) Undersampling with NearMiss
b) Oversampling with SMOTE

IV. **Topological properties importance**
a) Testing without topological properties
c) Testing including topological properties

IV. **Candidates to carry out further studies (in-vivo experiments)**
a) False positive analysis

Gather Sense of Our Data:

The first thing we must do is gather a **basic sense** of our data.

Summary:

- The total number of genes annotated is relatively **small**.
- There are no **"Null"** values, so we don't have to work on ways to replace values.
- Most of the genes were **Non-Annotated** (98.81%), while there are **Annotated** genes (1.19%).

In [0]:

```
# Imported Libraries
import io
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
from tqdm import tqdm #
import matplotlib.pyplot as plt # plotting
import seaborn as sns
import time
from google.colab import files

# Classifier Libraries
import xgboost as xgb

# Metrics Libraries
from sklearn.metrics import roc_curve, roc_auc_score, average_precision_score
from sklearn.metrics import balanced_accuracy_score, f1_score, accuracy_score

# Sampling and Validation Libraries
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import StratifiedShuffleSplit, train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import KFold, StratifiedKFold
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss

# Other Libraries
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
import warnings
warnings.filterwarnings("ignore")
```

In [29]:
```
# Set target annotation
target_annot = '0006807' #'0045454' #'0006807' #'0006412'

# Load biological processes name
# uploaded = files.upload()
bp_names = pd.read_csv('OSA-GeneAnnotation_names(BP).csv', ':', header=None, name
s=["go", "process"], dtype={'go': object})

# Load funtional annotations and topological properties dataset
# uploaded = files.upload()
df = pd.read_csv('OSA-GeneAnnotation(BP).csv')
entrez_df = df['entrez']
df.drop(['entrez'], axis=1, inplace=True)
print(df.shape)
df.head()
```

```
(19665, 630)
```

Out[29]:

	0019509	0006457	0016226	0006508	0006810	0055085	0006096	0051205	0006418	0006355	0007165	000
0	0	0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 630 columns

In [30]:
```
df.describe()
```

Out[30]:

	0019509	0006457	0016226	0006508	0006810	0055085	0006096	
count	19665.000000	19665.000000	19665.000000	19665.000000	19665.000000	19665.00000	19665.000000	196
mean	0.000153	0.008238	0.000610	0.014340	0.013679	0.01968	0.002543	
std	0.012351	0.090391	0.024696	0.118892	0.116158	0.13890	0.050361	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000	1.000000	

8 rows × 629 columns

In [31]:
```
df.columns
```

Out[31]:
```
Index(['0019509', '0006457', '0016226', '0006508', '0006810', '0055085',
       '0006096', '0051205', '0006418', '0006355',
       ...
       'Radiality', 'Stress', 'TopologicalCoefficient',
       'BetweennessCentrality', 'NumberOfUndirectedEdges', 'SelfLoops',
       'IsSingleNode', 'NumberOfDirectedEdges', 'AverageShortestPathLength',
       'NeighborhoodConnectivity'],
      dtype='object', length=630)
```

In [32]:
```
# Good No Null Values!
df.isnull().sum().max()
```

Out[32]:
```
0
```

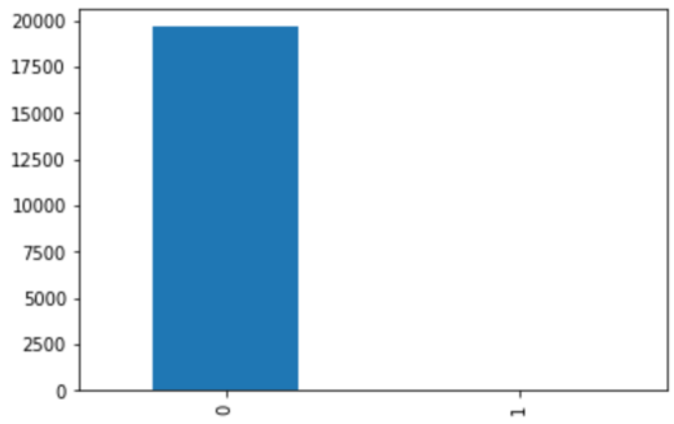
In [34]:
```
# The classes are heavily skewed we need to solve this issue later.
print(df[target_annot].value_counts())
print('No annot.', round(df[target_annot].value_counts()[0]/len(df) * 100,2), '% of
the dataset')
print('Annot.', round(df[target_annot].value_counts()[1]/len(df) * 100,2), '% of th
e dataset')
```

```
0    19650
1       15
Name: 0006807, dtype: int64
No annot. 99.92 % of the dataset
Annot. 0.08 % of the dataset
```

Note: Notice how imbalanced is our original dataset! Most of the genes are non-annotated. If we use this dataframe as the base for our predictive models and analysis we might get a lot of errors and our algorithms will probably overfit since it will "assume" that most genes are not annotated. But we don't want our model to assume, we want our model to detect patterns!

In [36]:
```
# Default colors
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#7f7f7
f', '#bcbd22', '#17becf']

# Plot distribution of the annotated and non annotated genes
# Code here ...

df[target_annot].value_counts().plot(kind='bar')
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff2612ddba8>
```



Distributions: By seeing the distributions we can have an idea how skewed are these features, we can also see further distributions of the other features. There are techniques that can help the distributions be less skewed which will be implemented in this notebook in the future.

Scaling and Correlation

In this phase of our kernel, we will first scale the columns related to the topological properties of the co-expression network. On the other hand, we need to also create a sub sample of the dataframe in order to have an equal amount of Annotated and Non-Annotated cases, helping our algorithms better understand patterns that determines whether a gene is involved in a biological process.

What is a sub-Sample?

In this scenario, our subsample will be a dataframe with a 50/50 ratio of annotated and non-annotated genes. Meaning our sub-sample will have the same amount of annotated and non annotated genes.

Why do we create a sub-Sample?

In the beginning of this notebook we saw that the original dataframe was heavily imbalanced! Using the original dataframe will cause the following issues:

- **Overfitting:** Our classification models will assume that in most cases there are not related to the biological process! What we want for our model is to be certain when a relation occurs.
- **Wrong Correlations:** It will be useful to understand how each of this features influence the result (Annotated or Non Annotated) by having an imbalance dataframe we are not able to see the true correlations between the class and features.

In [38]:
```
# We should scale the columns related to the topological properties of the co-expre
ssion network
# Dataset Features
topo_feature = ['ClosenessCentrality','Eccentricity','Degree','PartnerOfMultiEdgedN
odePairs','ClusteringCoefficient','Radiality','TopologicalCoefficient','Betweenness
Centrality','NumberOfUndirectedEdges','SelfLoops','IsSingleNode','NumberOfDirectedE
dges','AverageShortestPathLength','NeighborhoodConnectivity', 'Stress']
annots_feature = ['0019509','0006457','0016226','0006508','0006810','0055085','0006
096','0051205','0006418','0006355','0007165','0006396','0006189','0006099','0015977
','0015940','0055114','0016070','0006833','0000105','0008152','0000272','0005975','
0006265','0015979','0015995','0006779','0030001','0006605','0006886','0017038','000
6629','0015937','0045454','0006437','0042549','0006400','0006470','0006098','000657
3','0016117','0009765','0006412','0015976','0006364','0006367','0006468','0009234
','0006855','0009058','0071951','0006520','0009088','0071805','0006821','0000917','
0006436','0007205','0048544','0016075','0006535','0006139','0015031','0006631','000
6259','0006281','0003333','0010024','0000160','0006614','0002098','0008033','000666
2','0006221','0044237','0006424','0043039','0000154','0008652','0009165','0009116
','0009156','0044249','0006505','0018160','0033014','0015992','0008219','0006351','
0031123','0043631','0006979','0006777','0009813','0042398','0009231','0006814','000
9082','0006730','0006435','0009785','0006817','0006487','0006414','0006430','000680
7','0009107','0030163','0008299','0071722','0006465','0051188','0015986','0006427
','0009987','0051252','0009308','0006796','0006428','0043086','0006289','0006413','
0009408','0006790','0006069','0006729','0016559','0009089','0006006','0009416','004
5038','0006857','0001522','0009451','0006749','0006754','0006812','0006353','000643
1','0006801','0019538','0006415','0006352','0042254','0009094','0009584','0009585
','0017006','0018106','0018298','0006163','0015969','0006788','0016485','0070588','
0006260','0009306','0006450','0022900','0009228','0006164','0032957','0005978','003
1167','0006544','0006563','0006108','0044262','0009052','0032259','0032968','004423
8','0006334','0006426','0006633','0006568','0006464','0006597','0008295','0009081
','0006546','0006542','0016480','0006298','0045005','0006694','0009396','0006000','
0006003','0000079','0051726','0008643','0009168','0030494','0006783','0006887','000
6824','0009236','0016114','0006223','0006811','0006541','0006537','0042026','004426
7','0006388','0006014','0006073','0006419','0006952','0016310','0006595','0006564
','0006571','0042218','0006438','0006433','0009435','0006402','0006072','0046168','
0019752','0008610','0006168','0006421','0010027','0042558','0019430','0009252','001
5684','0046939','0006184','0009073','0015746','0006950','0006012','0006423','000676
0','0042823','0008615','0009439','0007050','0008654','0019464','0008360','0009273
','0051301','0006310','0006974','0006869','0043043','0048268','0019836','0007067','
0043412','0006434','0006750','0006813','0006432','0006165','0006183','0006228','000
6241','0034755','0006506','0006200','0006308','0007018','0007264','0042545','000972
5','0006913','0030244','0006284','0007017','0051258','0009664','0016567','0046274
','0006725','0032312','0006486','0006081','0019953','0005985','0000103','0035556','
0006071','0008283','0006644','0016042','0006536','0016043','0030036','0009607','003
0833','0006559','0000077','0000226','0071577','0051013','0010215','0016049','001631
1','0016192','0006102','0007275','0006354','0006357','0032784','0005992','0046488
','0045892','0048193','0006511','0006066','0009247','0030259','0006032','0016998','
0006909','0006278','0010338','0009086','0016125','0015991','0045116','0006820','004
4070','0006888','0009611','0006499','0019856','0006879','0051603','0018279','000656
1','0006422','0046417','0043085','0006635','0006461','0032012','0006596','0072488
','0046836','0006166','0030042','0006626','0045039','0006591','0006207','0006222','
0042546','0006275','0007021','0006566','0008272','0015671','0006526','0042450','003
0418','0022904','0006106','0017183','0007186','0000902','0007010','0006302','000645
2','0008612','0045901','0045905','0006122','0007030','0006865','0019318','0045900
','0007047','0006744','0019307','0006808','0006425','0006621','0046034','0006333','
0030071','0031145','0019673','0006090','0015780','0010315','0001510','0007155','004
2256','0042255','0009966','0006665','0006680','0016575','0043161','0009102','005118
6','0042176','0045980','0006479','0006825','0035434','0006897','0009405','0031120
','0042742','0050832','0034968','0007034','0006839','0006397','0009072','0009443','
0009909','0048573','0015743','0005986','0009690','0006471','0015770','0046470','001
7148','0006306','0090116','0006527','0015074','0006904','0010044','0009790','000609
4','0016068','0009873','0009269','0019419','0032313','0006021','0009415','0016458
','0006606','0051103','0009250','0006208','0044205','0032065','0010029','0016973','
0043666','0006370','0019370','0000162','0046835','0019358','0007585','0006446','003
0488','0032324','0045893','0046907','0006379','0009103','0007154','0000398','000704
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in get_loc(se
lf, key, method, tolerance)
   2896             try:
-> 2897                 return self._engine.get_loc(key)
   2898             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTa
ble.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTa
ble.get_item()

KeyError: 'ClosenessCentrality'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-38-ca47f33fc0e8> in <module>()
      5
      6 for fn in topo_feature:
----> 7   df['scaled_{0}'.format(fn)] = rob_scaler.fit_transform(df[fn].values.r
eshape(-1,1))
      8   df.drop([fn], axis=1, inplace=True)

/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py in __getitem__(self,
key)
   2993             if self.columns.nlevels > 1:
   2994                 return self._getitem_multilevel(key)
-> 2995             indexer = self.columns.get_loc(key)
   2996             if is_integer(indexer):
   2997                 indexer = [indexer]

/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in get_loc(se
lf, key, method, tolerance)
   2897                 return self._engine.get_loc(key)
   2898             except KeyError:
-> 2899                 return self._engine.get_loc(self._maybe_cast_indexer(ke
y))
   2900         indexer = self.get_indexer([key], method=method, tolerance=toler
ance)
   2901         if indexer.ndim > 1 or indexer.size > 1:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTa
ble.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTa
ble.get_item()

KeyError: 'ClosenessCentrality'
```

In [39]:
```
i = 0
for fn in topo_feature:
  scaled_feature = df['scaled_{0}'.format(fn)]
  df.drop(['scaled_{0}'.format(fn)], axis=1, inplace=True)
  df.insert(i, 'scaled_{0}'.format(fn), scaled_feature)
  i += 1

# Features are Scaled!
df.head()
```

Out[39]:

	scaled_ClosenessCentrality	scaled_Eccentricity	scaled_Degree	scaled_PartnerOfMultiEdgedNodePairs	scalec
0	0.340836	-1.0	0.277778	0.0	
1	-0.326122	-1.0	0.722222	0.0	
2	-0.279151	-1.0	0.203704	0.0	
3	0.203376	0.0	2.407407	0.0	
4	0.645119	0.0	1.129630	0.0	

5 rows × 630 columns

Note: Notice that the topological properties have been scaled, i.e., all its values are within the range $[-1, 1]$.

In [0]:
```
scaled_topo_feature = ['scaled_{0}'.format(fn) for fn in topo_feature]

# Plot correlation heatmap between topological properties
# Code here ...
```

Correlation: Notice that some topological properties strongly correlated. This features does not help the models in the prediction, therefore it is convenient to remove them from the dataset (remove noise)

In [0]:
```
for fn in ['PartnerOfMultiEdgedNodePairs','Radiality','NumberOfUndirectedEdges','Se
lfLoops','IsSingleNode','NumberOfDirectedEdges','AverageShortestPathLength','Stress
']:
  df.drop(['scaled_{0}'.format(fn)], axis=1, inplace=True)

topo_feature = ['ClosenessCentrality','Eccentricity','Degree','ClusteringCoefficien
t','TopologicalCoefficient','BetweennessCentrality','NeighborhoodConnectivity']
scaled_topo_feature = ['scaled_{0}'.format(fn) for fn in topo_feature]
```

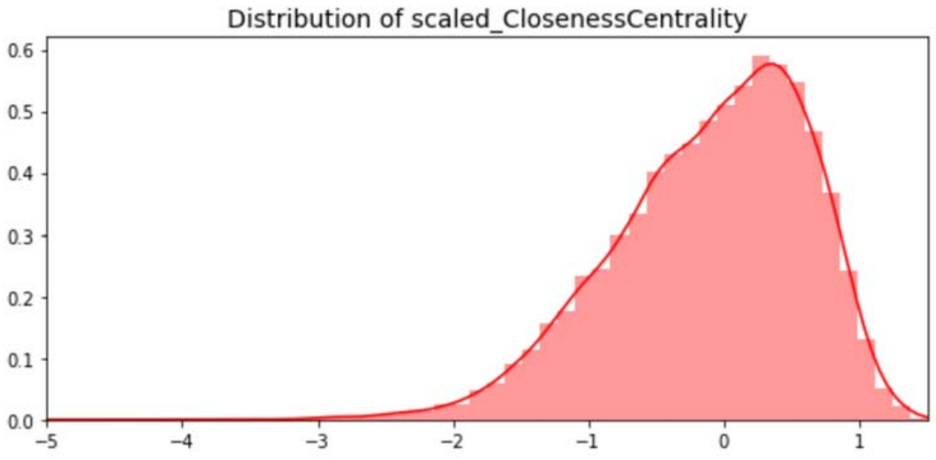
In [43]:
```
# Plot correlation heatmap between topological properties remaining
# Code here ...

df.describe()
```

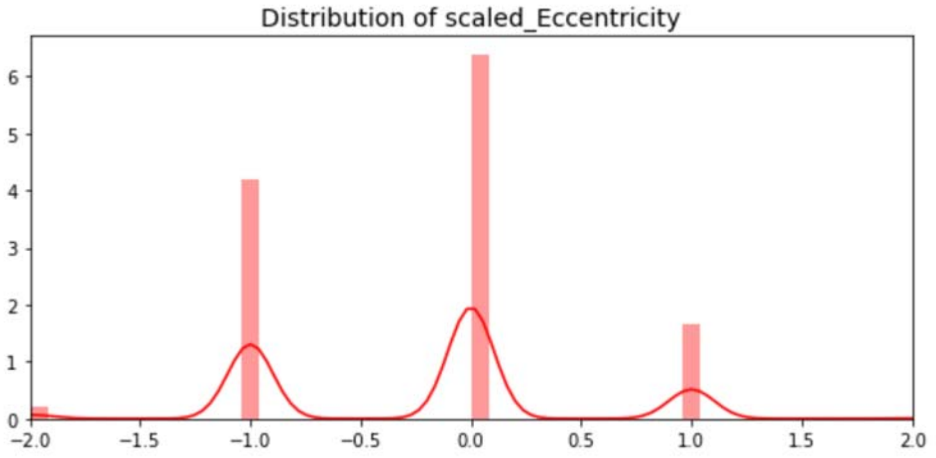
Out[43]:

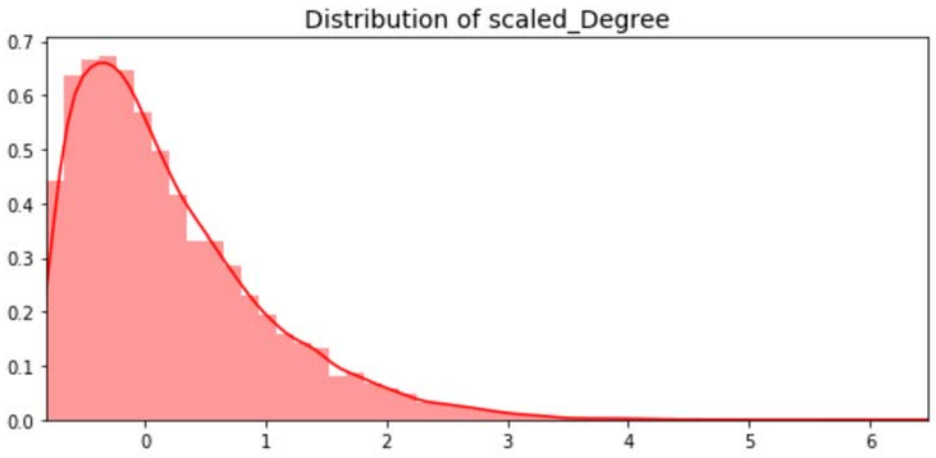
	scaled_ClosenessCentrality	scaled_Eccentricity	scaled_Degree	scaled_ClusteringCoefficient	scaled_Top
count	19665.000000	19665.000000	19665.000000	19665.000000	
mean	-0.098712	-0.233969	0.206086	0.187713	
std	0.716488	0.701638	0.791313	0.892680	
min	-5.002906	-2.000000	-0.814815	-1.480227	
25%	-0.552322	-1.000000	-0.388889	-0.416327	
50%	0.000000	0.000000	0.000000	0.000000	
75%	0.447678	0.000000	0.611111	0.583673	
max	1.510494	2.000000	6.481481	4.678760	

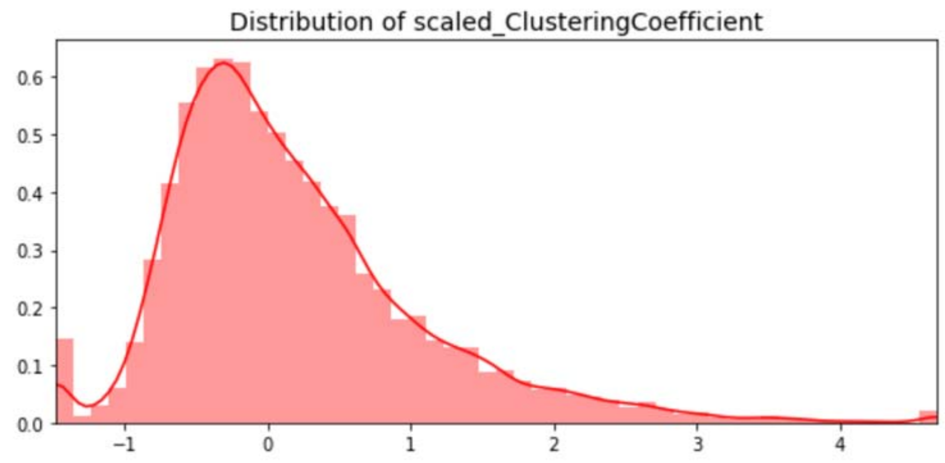
8 rows × 622 columns

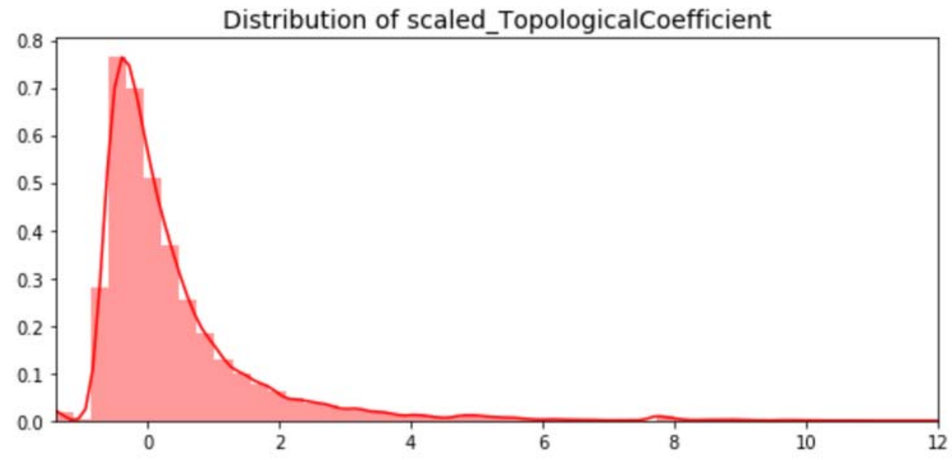
In [42]:
```
for fn in scaled_topo_feature:
  fig, ax = plt.subplots(figsize=(9,4))

  feature_val = df[fn].values

  sns.distplot(feature_val, ax=ax, color='r')
  ax.set_title('Distribution of {0}'.format(fn), fontsize=14)
  ax.set_xlim([min(feature_val), max(feature_val)])

  plt.show()
  plt.close()
```

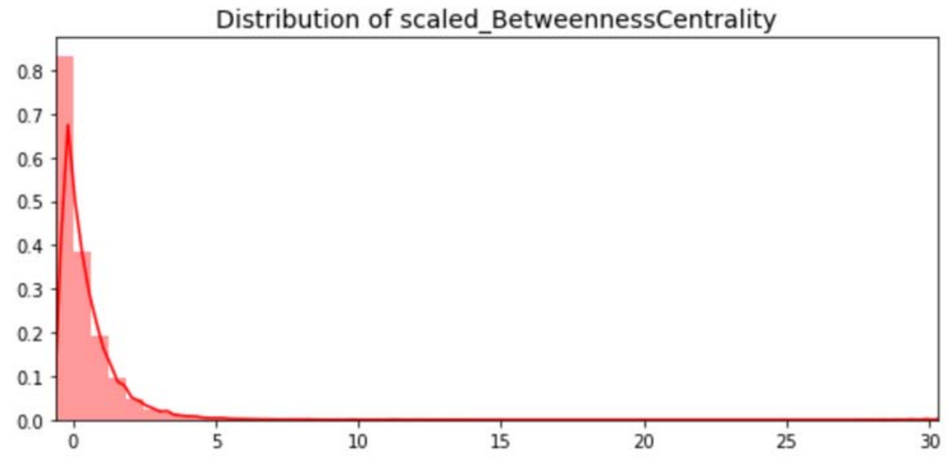

Distribution of scaled_ClosenessCentrality
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−5
−4
−3
−2
−1
0
1

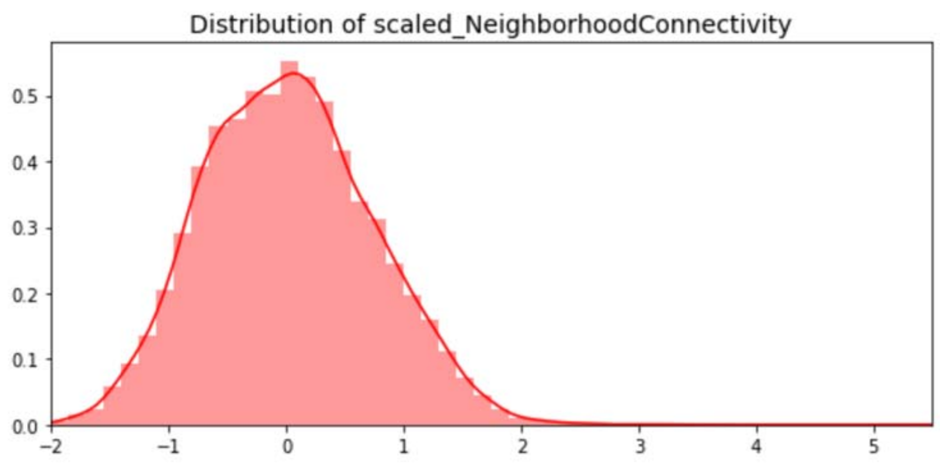

Distribution of scaled_Eccentricity
6
5
4
3
2
1
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Distribution of scaled_Degree
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5
6


Distribution of scaled_ClusteringCoefficient


Distribution of scaled_TopologicalCoefficient


Distribution of scaled_BetweennessCentrality



Undersampling vs. Oversampling

Splitting the Data (Original DataFrame)

Before proceeding with the **Random UnderSampling technique** we have to separate the orginal dataframe. **Why? for testing purposes, remember although we are splitting the data when implementing Random UnderSampling or OverSampling techniques, we want to test our models on the original testing set not on the testing set created by either of these techniques.** The main goal is to fit the model either with the dataframes that were undersample and oversample (in order for our models to detect the patterns), and test it on the original testing set.

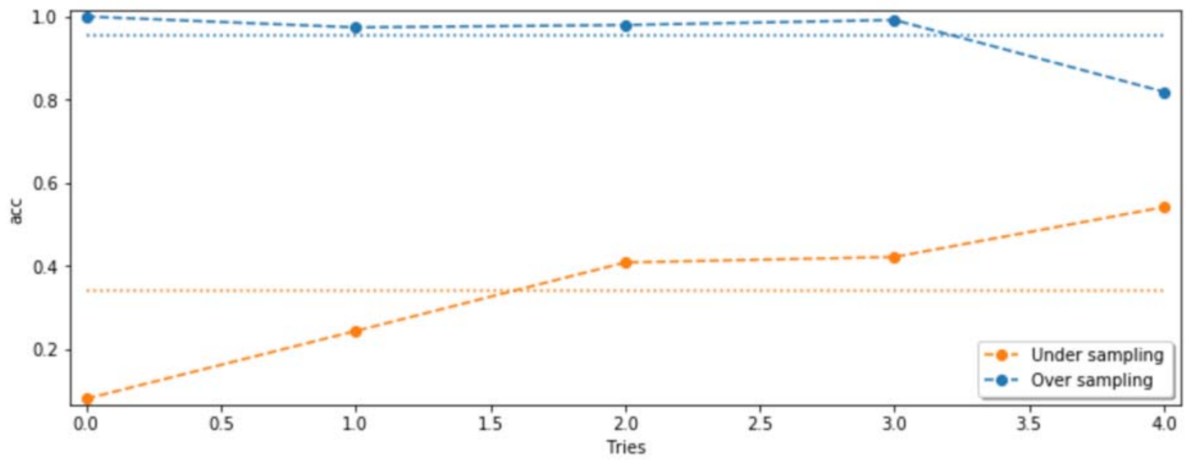
In [0]:
```
# Data structure to store the metrics
xgb_under = {'acc':list(), 'bac':list(), 'f1s':list(), 'roc':list(), 'avp':list()}
xgb_over = {'acc':list(), 'bac':list(), 'f1s':list(), 'roc':list(), 'avp':list()}
```

In [44]:
```
t0 = time.time()
target_name = bp_names.loc[bp_names['go']==target_annot].values[0]

#################
# Splitting the Data (Original DataFrame)
#################
X = df.drop(target_annot, axis=1)
y = df[target_annot]
sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in sss.split(X, y):
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values
# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_counts=T
rue)
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=Tru
e)

# Use RandomizedSearchCV to find the best parameters.
# GradientBoosting Classifier
xgb_params = {"max_depth": list(range(2,5,1)), "n_estimators": list(range(1,5,1)),
              "min_samples_leaf": list(range(5,7,1)), 'colsample_bytree': list(np.a
range(0.1, 1.1, 0.1))}
rand_xgb = RandomizedSearchCV(xgb.XGBClassifier(nthread=-1, random_state=2019), xgb
_params, n_iter=4)

#################
# Under-Sampling
#################
# List to append the score and then find the average
undersample_accuracy, undersample_balancedacc = list(), list()
undersample_f1, undersample_auc, undersample_average_precision = list(), list(), li
st()

# Cross Validating the right way
for train, test in tqdm(sss.split(original_Xtrain, original_ytrain)):
  undersample_pipeline = imbalanced_make_pipeline(NearMiss(sampling_strategy='major
ity'), rand_xgb)
  undersample_model = undersample_pipeline.fit(original_Xtrain[train], original_ytr
ain[train])
  undersample_prediction = undersample_model.predict(original_Xtrain[test])

  undersample_accuracy.append(undersample_pipeline.score(original_Xtrain[test], ori
ginal_ytrain[test]))
  undersample_balancedacc.append(balanced_accuracy_score(original_ytrain[test], und
ersample_prediction))
  undersample_f1.append(f1_score(original_ytrain[test], undersample_prediction))
  undersample_average_precision.append(average_precision_score(original_ytrain[tes
t], undersample_prediction))
  undersample_auc.append(roc_auc_score(original_ytrain[test], undersample_predictio
n))

xgb_under['acc'] = list(undersample_accuracy)
xgb_under['bac'] = list(undersample_balancedacc)
xgb_under['f1s'] = list(undersample_f1)
xgb_under['roc'] = list(undersample_auc)
xgb_under['avp'] = list(undersample_average_precision)

#################
```

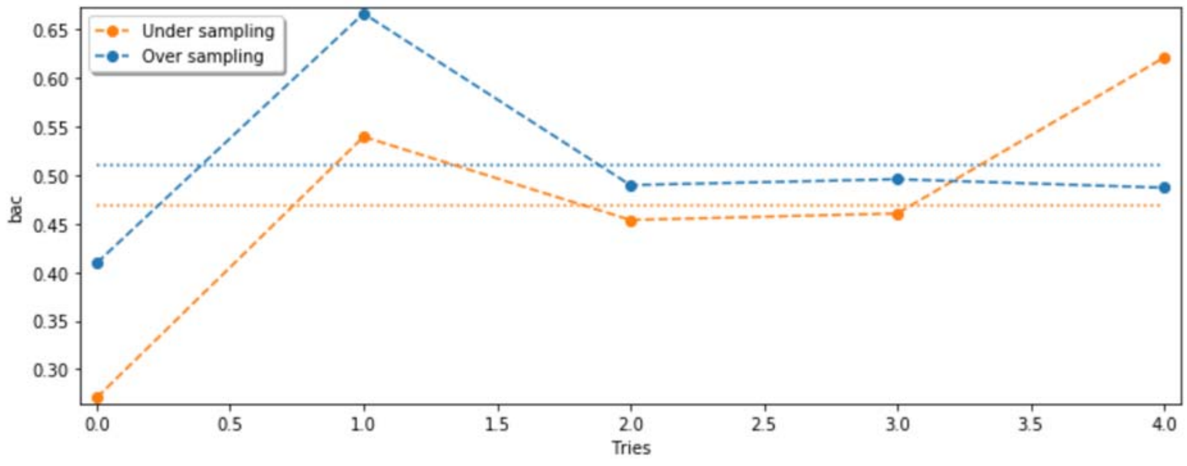
```
5it [00:01,  2.76it/s]
5it [00:50, 10.20s/it]

Took 5.3e+01 s
```

In [45]:
```
##
## Comparing metrics for the models
## Sorted by difference of models (under vs over)
##
measures = ['acc','bac','f1s','roc','avp']
for ms in measures:
  y_diff, y_1, y_2 = list(), list(), list()
  c = 0
  for i in range(len(xgb_under[ms])):
    y1 = xgb_under[ms][i]
    y2 = xgb_over[ms][i]
    y_diff.append((y2 - y1, y1, y2))
    if y2 - y1 > 0: c += 1
  y_diff.sort(reverse=True)

  print('{0}: oversampling is better than undersampling {1} times ({2:.2f}%)'.forma
t(ms, c,(c*100)/len(xgb_under[ms])))

  fig, ax = plt.subplots(figsize=(10,4))
  plt.plot(range(len(y_diff)), [y[1] for y in y_diff], 'o--', color=colors[1], labe
l='Under sampling')
  plt.plot(range(len(y_diff)), [y[2] for y in y_diff], 'o--', color=colors[0], labe
l='Over sampling')
  plt.hlines(np.mean(xgb_under[ms]), xmin=0, xmax=len(xgb_under[ms])-1, color=color
s[1], linestyle='dotted')
  plt.hlines(np.mean(xgb_over[ms]), xmin=0, xmax=len(xgb_over[ms])-1, color=colors
[0], linestyle='dotted')
  plt.legend(loc='best', shadow=True, fontsize='medium')
  plt.margins(0.015)
  plt.xlabel('Tries')
  plt.ylabel(ms)
  plt.tight_layout()
  plt.show()
  # plt.savefig('{0}_sampling.eps'.format(ms), format='eps', dpi=600)
  plt.close()
```

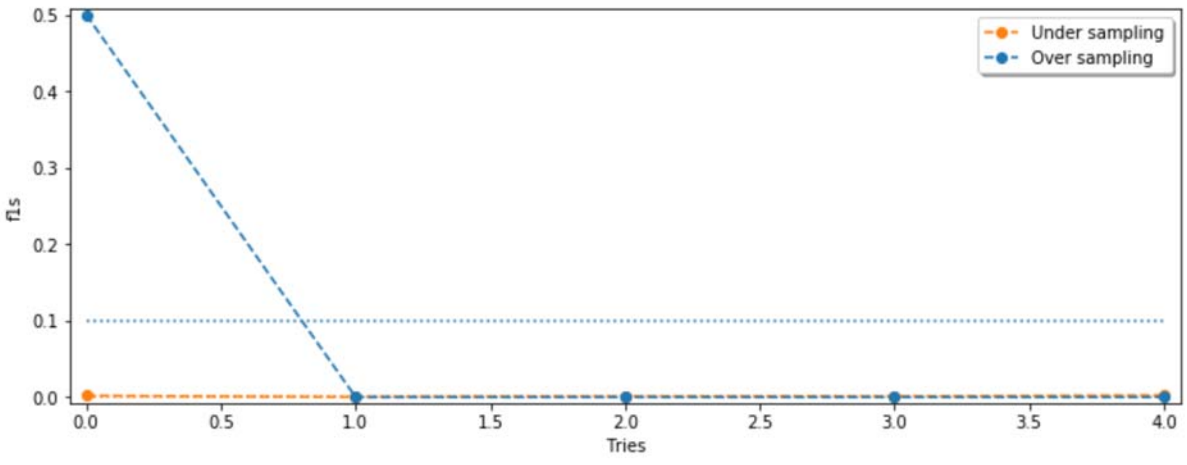
```
acc: oversampling is better than undersampling 5 times (100.00%)
```



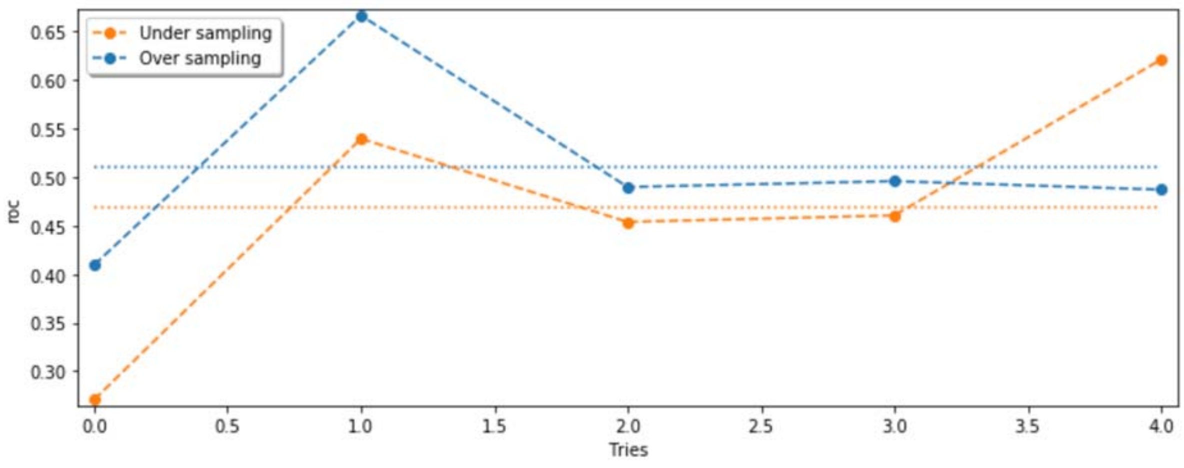
```
bac: oversampling is better than undersampling 4 times (80.00%)
```



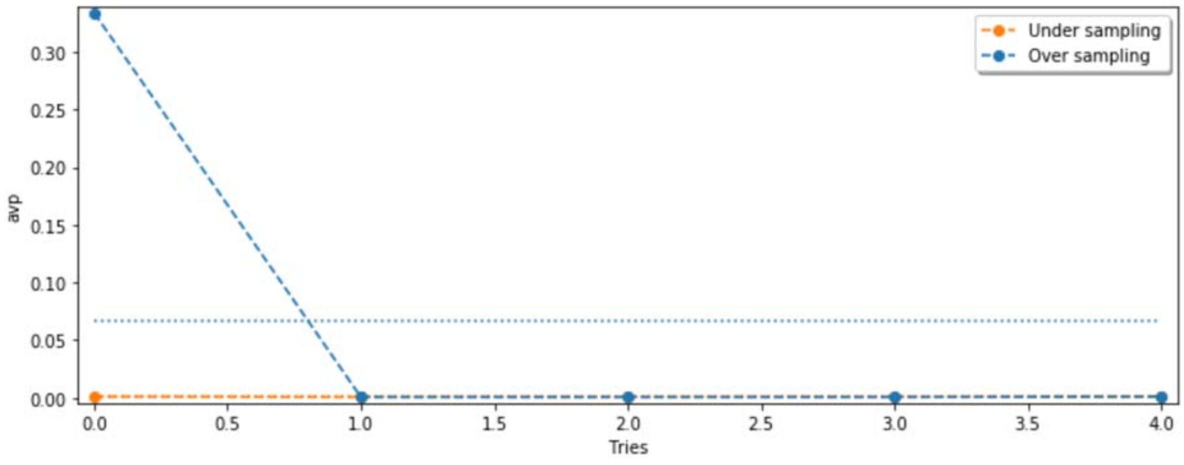
```
f1s: oversampling is better than undersampling 1 times (20.00%)
```



```
roc: oversampling is better than undersampling 4 times (80.00%)
```



```
avp: oversampling is better than undersampling 3 times (60.00%)
```



Topological properties

We should determine if the topological properties help in obtaining a more precise prediction for annotating genes.

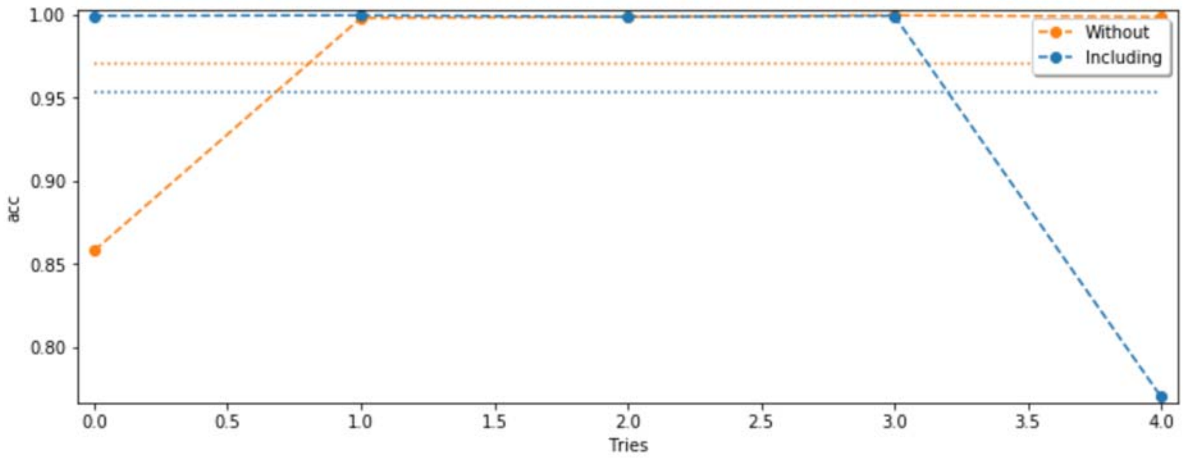
In [0]:
```
# Data structure to store the metrics
xgb_over_annot = {'acc':list(), 'bac':list(), 'f1s':list(), 'roc':list(), 'avp':lis
t()}
xgb_over_topm = {'acc':list(), 'bac':list(), 'f1s':list(), 'roc':list(), 'avp':list
()}
```

In [47]:
```
t0 = time.time()
target_name = bp_names.loc[bp_names['go']==target_annot].values[0]

# Classifier with optimal parameters
xgb_params = {"max_depth": list(range(2,5,1)), "n_estimators": list(range(1,5,1)),
              "min_samples_leaf": list(range(5,7,1)), 'colsample_bytree': list(np.a
range(0.1, 1.1, 0.1))}
rand_xgb = RandomizedSearchCV(xgb.XGBClassifier(nthread=-1, random_state=2019), xgb
_params, n_iter=4)

################
# Dataset without topological properties
################
# Splitting the Data (Annotations)
X = df[annots_feature]
X = df.drop(target_annot, axis=1)
y = df[target_annot]
sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
for train_index, test_index in sss.split(X, y):
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values
# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_counts=T
rue)
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=Tru
e)

# Implementing SMOTE Technique
# Cross Validating the right way
oversample_accuracy, oversample_balancedacc = list(), list()
oversample_f1, oversample_auc, oversample_average_precision = list(), list(), list
()
for train, test in tqdm(sss.split(original_Xtrain, original_ytrain)):
    # SMOTE happens during Cross Validation not before..
    pipeline = imbalanced_make_pipeline(SMOTE(sampling_strategy='minority'), rand_x
gb)
    model = pipeline.fit(original_Xtrain[train], original_ytrain[train])
    best_est = rand_xgb.best_estimator_
    prediction = best_est.predict(original_Xtrain[test])

    oversample_accuracy.append(pipeline.score(original_Xtrain[test], original_ytrai
n[test]))
    oversample_balancedacc.append(balanced_accuracy_score(original_ytrain[test], pr
ediction))
    oversample_f1.append(f1_score(original_ytrain[test], prediction))
    oversample_average_precision.append(average_precision_score(original_ytrain[tes
t], prediction))
    oversample_auc.append(roc_auc_score(original_ytrain[test], prediction))

xgb_over_annot['acc'] = list(oversample_accuracy)
xgb_over_annot['bac'] = list(oversample_balancedacc)
xgb_over_annot['f1s'] = list(oversample_f1)
xgb_over_annot['avp'] = list(oversample_average_precision)
xgb_over_annot['roc'] = list(oversample_auc)

################
# Dataset including topological properties
################
# Splitting the Data (Annotations)
```

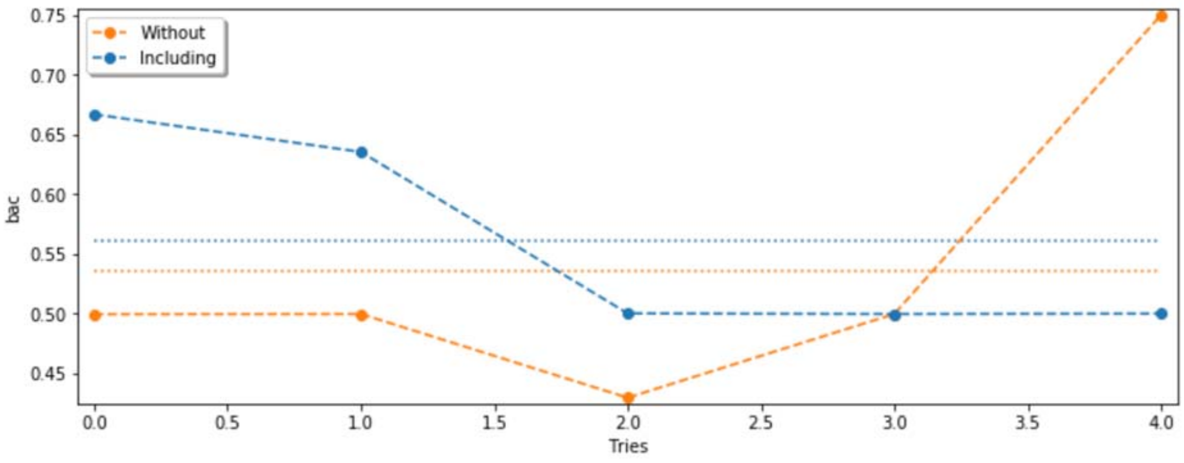
```
5it [00:50, 10.24s/it]
5it [00:50,  9.99s/it]

Took 1e+02 s
```

In [19]:
```
##
## Comparing performance metrics for the models
## Sorted by difference of models (with and without topological properties)
##
measures = ['acc', 'bac', 'f1s', 'roc', 'avp']
for ms in measures:
  y_diff, y_1, y_2 = list(), list(), list()
  c = 0
  for i in range(len(xgb_over_annot[ms])):
    y1 = xgb_over_annot[ms][i]
    y2 = xgb_over_topm[ms][i]
    y_diff.append((y2 - y1, y1, y2))
    if y2 - y1 > 0: c += 1
  y_diff.sort(reverse=True)

  print('{0}: Including topological properties is better {1} times ({2:.2f})'.forma
t(ms, c, (c*100)/len(y_diff)))

  fig, ax = plt.subplots(figsize=(10,4))
  plt.plot(range(len(y_diff)), [y[1] for y in y_diff], 'o--', label='Without', colo
r=colors[1])
  plt.plot(range(len(y_diff)), [y[2] for y in y_diff], 'o--', label='Including', co
lor=colors[0])
  plt.hlines(np.mean(xgb_over_annot[ms]), xmin=0, xmax=len(xgb_under[ms])-1, color=
colors[1], linestyle='dotted')
  plt.hlines(np.mean(xgb_over_topm[ms]), xmin=0, xmax=len(xgb_over[ms])-1, color=co
lors[0], linestyle='dotted')
  plt.legend(loc='best', shadow=True, fontsize='medium')
  plt.margins(0.015)
  plt.xlabel('Tries')
  plt.ylabel(ms)
  plt.tight_layout()
  plt.show()
  # plt.savefig('{0}_topological.eps'.format(ms), format='eps', dpi=600)
  plt.close()
```

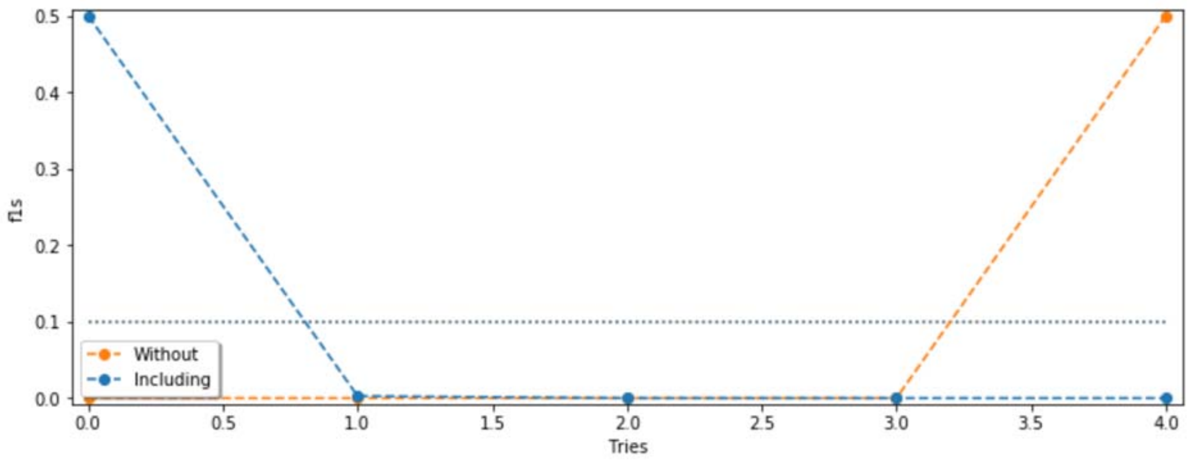
```
acc: Including topological properties is better 2 times (40.00)
```



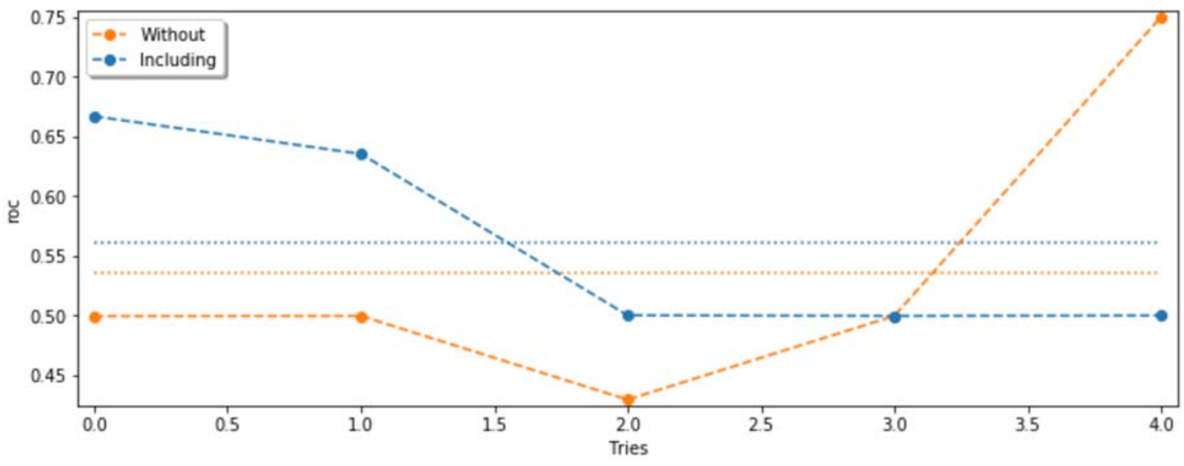
```
bac: Including topological properties is better 3 times (60.00)
```



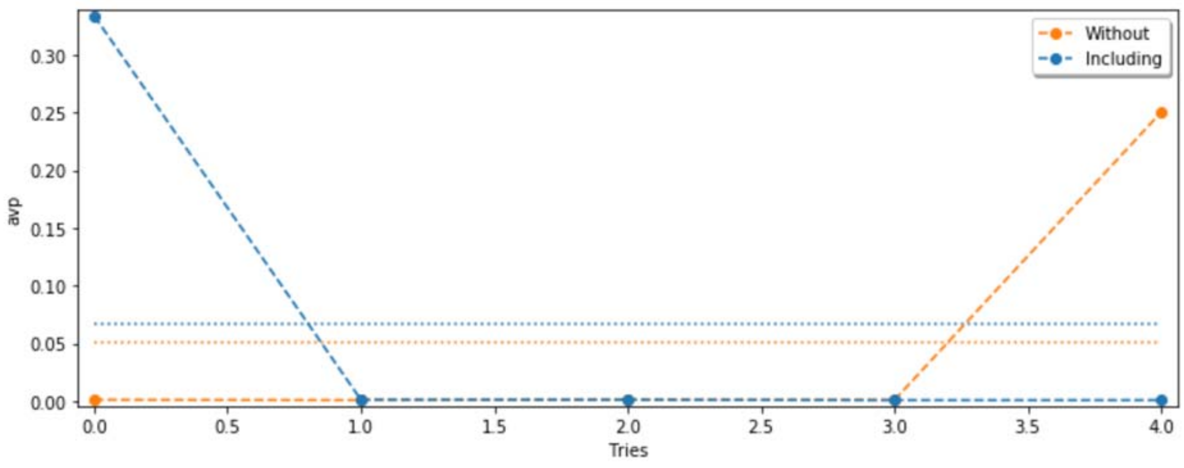
```
f1s: Including topological properties is better 2 times (40.00)
```



```
roc: Including topological properties is better 3 times (60.00)
```



```
avp: Including topological properties is better 2 times (40.00)
```



Candidates to carry out further studies (in-vivo experiments)

A false positive analysis is applied to the annotation predictions: the idea is to identify the genes that tend to be classified as a false positive because they are the candidate genes on which lab experimentation can focus on to discover unknown annotations.

In [0]:
```
xgb_over_trg = {'acc':list(), 'bac':list(), 'f1s':list(), 'roc':list(), 'avp':list
()}
```

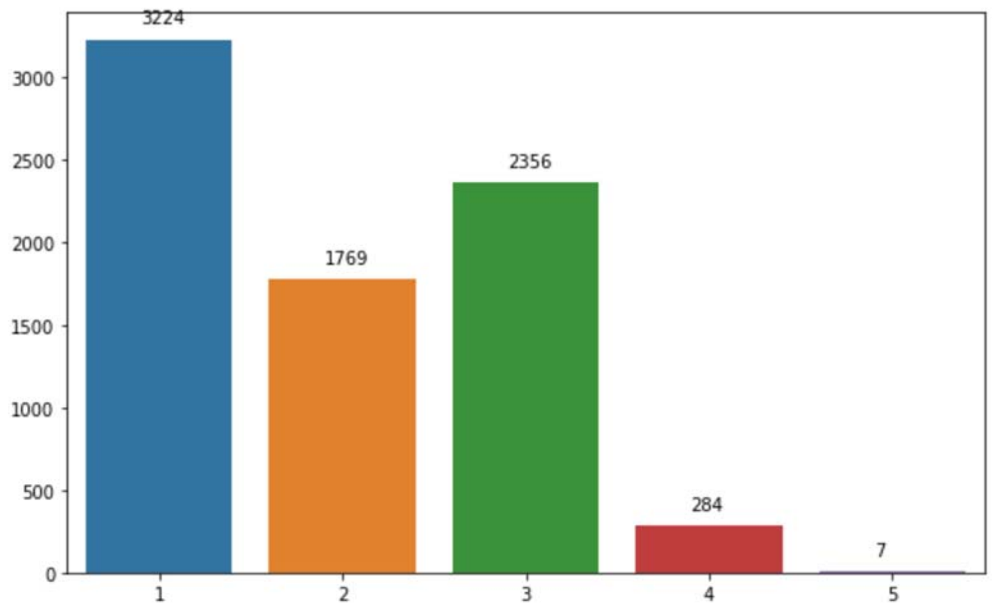
```
In [21]: t0 = time.time()
target_name = bp_names.loc[bp_names['go']==target_annot].values[0]
fp_genes = list()

################
# Splitting the Data (Original DataFrame)
################
X = df.drop(target_annot, axis=1)
y = df[target_annot]
sss = StratifiedKFold(n_splits=10, random_state=None, shuffle=False)
for train_index, test_index in sss.split(X, y):
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values

################
# SMOTE Technique (Over-Sampling)
################
# Classifier with optimal parameters
xgb_params = {"max_depth": list(range(2,5,1)), "n_estimators": list(range(1,5,1)),
              "min_samples_leaf": list(range(5,7,1)), 'colsample_bytree': list(np.a
range(0.1, 1.1, 0.1))}
rand_xgb = RandomizedSearchCV(xgb.XGBClassifier(nthread=-1, random_state=2019), xgb
_params, n_iter=4)

# Implementing SMOTE Technique
# Cross Validating the right way
accuracy_lst, balancedacc_lst, average_precision_lst, f1_lst, auc_lst = list(), lis
t(), list(), list(), list()
for train, test in tqdm(sss.split(original_Xtrain, original_ytrain)):
    pipeline = imbalanced_make_pipeline(SMOTE(sampling_strategy='minority'), rand_x
gb) # SMOTE happens during Cross Validation not before..
    model = pipeline.fit(original_Xtrain[train], original_ytrain[train])
    best_est = rand_xgb.best_estimator_
    prediction = best_est.predict(original_Xtrain)
    _tmp = ((original_ytrain == 0) & (prediction == 1))
    _tmp = np.where(_tmp==True)[0].tolist()
    fp_genes += _tmp

    accuracy_lst.append(pipeline.score(original_Xtrain, original_ytrain))
    balancedacc_lst.append(balanced_accuracy_score(original_ytrain, prediction))
    f1_lst.append(f1_score(original_ytrain, prediction))
    average_precision_lst.append(average_precision_score(original_ytrain, predictio
n))
    auc_lst.append(roc_auc_score(original_ytrain, prediction))

xgb_over_trg['acc'] = list(accuracy_lst)
xgb_over_trg['bac'] = list(balancedacc_lst)
xgb_over_trg['f1s'] = list(f1_lst)
xgb_over_trg['roc'] = list(auc_lst)
xgb_over_trg['avp'] = list(average_precision_lst)

t1 = time.time()
print("Took {:.2} s".format(t1 - t0))
```

```
10it [02:06, 12.68s/it]

Took 1.3e+02 s
```

False Positive: A gene that is consistenly annotated for the ML model although it is included in the body of knowledge, is called a false positive.

In [22]:
```
# False positive (fp) genes
fp_df = pd.DataFrame(fp_genes, columns=[target_annot])

# Count fp genes frequency
fp_vc = pd.DataFrame(fp_df[target_annot].value_counts(dropna=True, sort=True).reset
_index())
gene_count = fp_vc.shape[0]
fp_vc.columns = ['gene', 'counts']
fp_vc['gene'] = pd.to_numeric(fp_vc['gene'], downcast='integer')
mfp_genes = fp_vc['counts'].max()

# Replace gene index with gene entrez id
vc = fp_vc[fp_vc['counts'] == mfp_genes]
fp_genes_names = list()
for g in vc['gene'].tolist():
  gname = entrez_df.loc[g]
  fp_genes_names.append(gname)
vc['gene'] = fp_genes_names

# Print and plot frequency information
print('{0}\nAnnotation: {1} ({2})\nTotal fp: \t   {3}\nTotal unique fp:   {4}\nMost
frequents fp: {5}\n'.format('#'*10 ,bp_names.loc[bp_names['go']==target_annot].valu
es[0][1].strip(), target_annot,
                                       len(fp_genes), len(fp_df[target_annot].uniqu
e()), vc.shape[0]))
print(fp_vc['counts'].value_counts())
fig, ax = plt.subplots(figsize=(8,5))
fp_freq = fp_vc['counts'].value_counts()# .plot(kind='bar')
sns.barplot(fp_freq.index, fp_freq.values)
for i,j in zip(fp_freq.index, fp_freq.values):
  ax.annotate(str(j),xy=(i-1.1,j+100))
plt.tight_layout()
plt.show()
```

```
##########
Annotation: nitrogen compound metabolic process (0006807)
Total fp:          15001
Total unique fp:   7640
Most frequents fp: 7

1    3224
3    2356
2    1769
4     284
5       7
Name: counts, dtype: int64
```



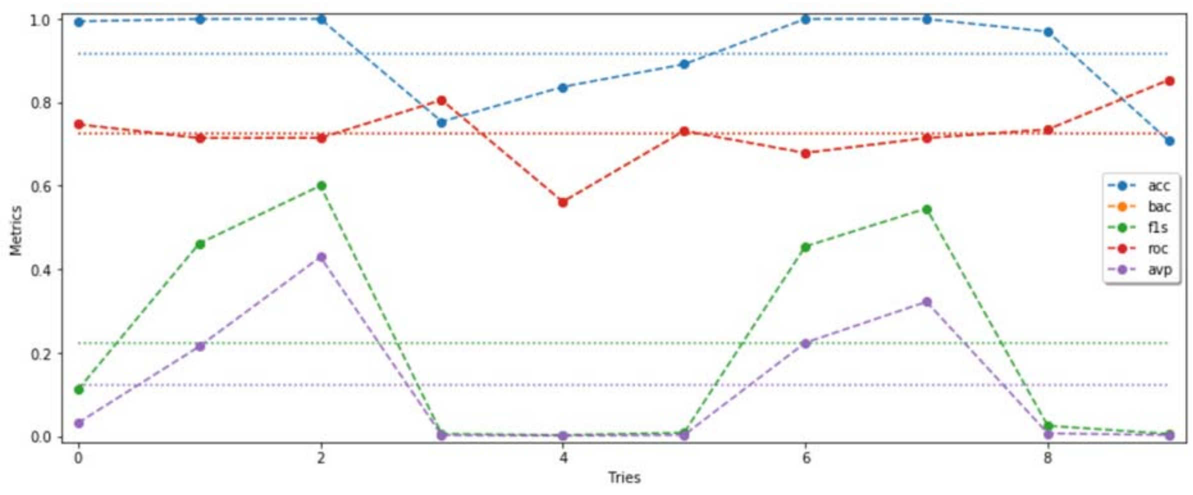
Candidates: We should focus on the genes most frequently annotated. The more experiments are done, the more precise the prediction is.

In [27]:
```
vc
```

Out[27]:

	gene	counts
0	4351459	5
1	4341252	5
2	4339989	5
3	4329369	5
4	4340970	5
5	4349318	5
6	4330040	5

In [24]:
```
# Plot performance metrics for the models in the false positive analysis
# Code here ...
```



References:

- Romero M., Finke J., Quimbaya M., Rocha C. (2020) In-silico Gene Annotation Prediction Using the Co-expression Network Structure. In: Cherifi H., Gaito S., Mendes J., Moro E., Rocha L. (eds) Complex Networks and Their Applications VIII. COMPLEX NETWORKS 2019. Studies in Computational Intelligence, vol 882. Springer, Cham
- Credit Fraud || Dealing with Imbalanced Datasets by Janio Martinez Bachmann. Kaggle

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

Author

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

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.

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.

Import libraries

In [0]:
```
import pandas as pd
import sys
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

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

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

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

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

In [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
```

Prepare data from RNA-seq

RNA-seq data was accessed through GEO database \cite{GEOAcces90:online} (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
(https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE98455)

Load Expression file

In [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)
```

Out[0]:

	GSM2596381_101_C_rep1	GSM2596382_101_C_rep2	GSM2596383_101_S_rep1	GSI
Gene				
13103.t02982	0.0	0.0	0.0	
13105.t01662	1.0	1.0	1.0	
13110.t02303	261.0	295.0	338.0	
13108.t00264	25.0	33.0	11.0	
13102.t01556	80.0	80.0	72.0	

5 rows × 368 columns

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

```
(10000, 368)
```

Out[0]:

	GSM2596381_101_C_rep1	GSM2596382_101_C_rep2	GSM2596383_101_S_rep1	GSI
Gene				
13108.t04075	138.0	171.0	130.0	
13104.t04582	3.0	14.0	4.0	
13108.t03131	68.0	67.0	66.0	
13110.t02328	0.0	0.0	0.0	
13101.t03278	0.0	0.0	0.0	

5 rows × 368 columns

DESeq normalization

In [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 (col
umn)
    medians = df_deseq.median(axis=0)
    # Step 6: Convert the medians to "normal numbers" to get the fina
l
    # 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
```

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

Out[0]:

	GSM2596381_101_C_rep1	GSM2596382_101_C_rep2	GSM2596383_101_S_rep1	GSI
Gene				
13108.t04075	162.709699	187.445151	127.649123	
13104.t04582	3.537167	15.346387	3.927665	
13108.t03131	80.175794	73.443422	64.806478	
13110.t02328	0.000000	0.000000	0.000000	
13101.t03278	0.000000	0.000000	0.000000	

5 rows × 368 columns

Average repetitions from each accession

In [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()
```

Out[0]:

	GSM2596381_101	GSM2596383_101	GSM2596385_105	GSM2596387_105	GSM259
Gene					
13108.t04075	175.077425	145.193552	135.247472	109.560777	13
13104.t04582	9.441777	5.965586	1.867674	6.155907	
13108.t03131	76.809608	68.419022	95.772150	69.133553	10
13110.t02328	0.000000	0.000000	0.000000	0.000000	
13101.t03278	0.000000	0.000000	0.000000	0.000000	

5 rows × 184 columns

Remove genes with low expression

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

In [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)
```

Remove genes with low variance:

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

In [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)
```

Separate Control and Stress data

In [0]:
```python
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()
```

Out[0]:

	GSM2596381_101	GSM2596385_105	GSM2596389_107	GSM2596393_109	GSM259
Gene					
13102.t00559	7.290879	12.067669	13.470515	73.763489	
13111.t00059	213.010304	111.580497	129.685364	46.632119	6
13112.t00015	19.339423	23.267933	28.644859	33.390887	
13104.t04551	43.054198	85.629922	37.815894	94.793739	4
13101.t05507	661.468112	690.949902	497.106770	857.850630	102

5 rows × 92 columns

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

Out[0]:

	GSM2596383_101	GSM2596387_105	GSM2596391_107	GSM2596395_109	GSM259
Gene					
13102.t00559	7.577417	9.927047	19.386431	56.984331	1
13111.t00059	200.254736	66.429743	65.158904	75.579547	5
13112.t00015	18.035716	16.594511	17.850433	35.729950	
13104.t04551	54.987205	87.205817	51.086096	47.376874	7
13101.t05507	669.897218	980.101515	613.821137	1153.678371	231

5 rows × 92 columns

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)

In [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)
```

Out[0]:

	GSM2596381	GSM2596385	GSM2596389	GSM2596393	GSM2596397	GSM259640
13102.t00559	0.049018	-0.258098	0.494493	-0.366671	0.363435	-0.84457
13111.t00059	-0.088658	-0.739500	-0.982090	0.685024	-0.084572	1.69672
13112.t00015	-0.095570	-0.463926	-0.653184	0.094931	1.206464	0.12802
13104.t04551	0.345818	0.026008	0.424251	-0.985614	0.700215	-0.91659
13101.t05507	0.018241	0.503735	0.303712	0.427012	1.174587	-0.21008

5 rows × 92 columns

Remove genes exhibiting low Log2Fold change variance

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

In [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)
```

Out[0]:

	GSM2596381	GSM2596385	GSM2596389	GSM2596393	GSM2596397	GSM259640
13102.t00559	0.049018	-0.258098	0.494493	-0.366671	0.363435	-0.84457
13111.t00059	-0.088658	-0.739500	-0.982090	0.685024	-0.084572	1.69672
13112.t00015	-0.095570	-0.463926	-0.653184	0.094931	1.206464	0.12802
13104.t04551	0.345818	0.026008	0.424251	-0.985614	0.700215	-0.91659
13101.t05507	0.018241	0.503735	0.303712	0.427012	1.174587	-0.21008

5 rows × 92 columns

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

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

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

Gene Module detection

WGCNA

The co-expression network is constructed using the R package WGCNA. **W**eighted **G**ene **C**oexpression **N**etwork **A**nalysis is a method of data mining widely used to study biological networks based on pairwise correlations between variables. It allows, among other things, to build the co-expression network and identify groups (modules) of highly correlated genes.

https://horvath.genetics.ucla.edu/html/CoexpressionNetwork/Rpackages/WGCNA/ (https://horvath.genetics.ucla.edu/html/CoexpressionNetwork/Rpackages/WGCNA/)

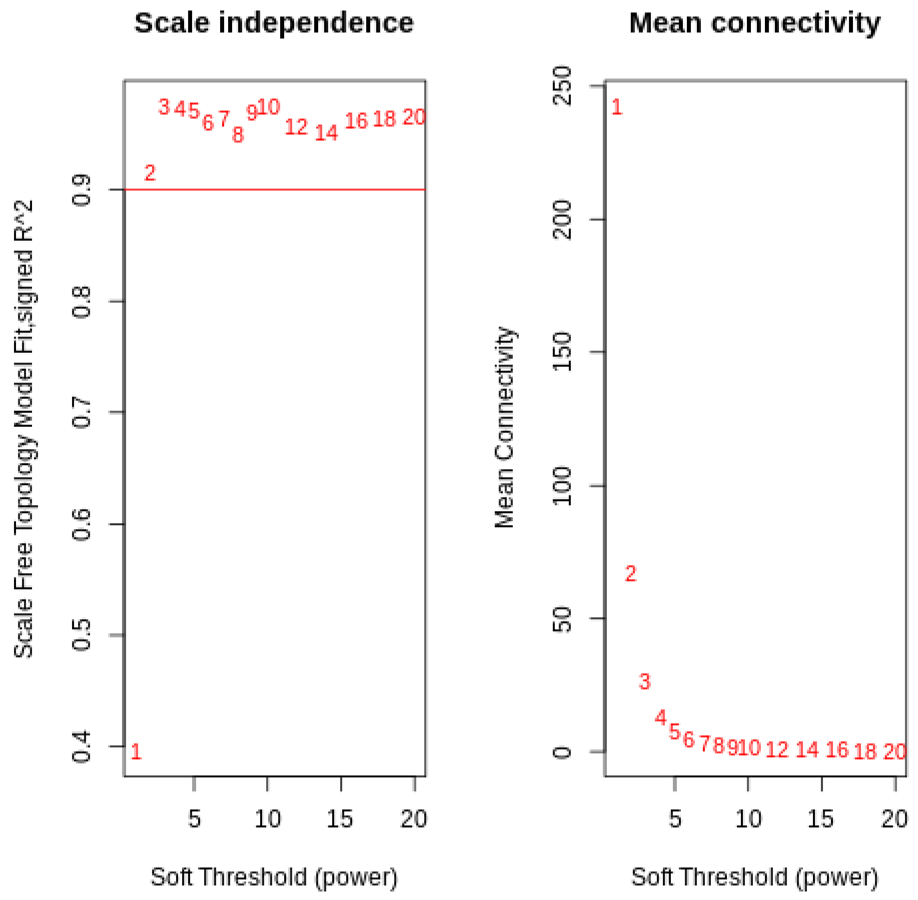
Network construction

- Similarity: $S = [s_{ij}]_{n \times n}$ measures the level of concordance between gene expression profiles across the experiments (absolute value of the Pearson correlation).
- Adjacency Function: $A = [a_{ij}]_{n \times n} = [(s_{ij})^{\beta}]_{n \times n}$ that encodes the connection strength between each pair of nodes (genes) and is computed as the similarity value up to a power $\beta > 1$ so the degree distribution will fit a small-word network.

In [0]:
```
%%R
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
```

In [0]:
```
%%R -i Log2FC
#-----------------------------------------------------------------------
-------
#Step-by-step network construction and module detection
#-----------------------------------------------------------------------
-------
# Choose a set of soft-thresholding powers
powers = c(c(1:10), seq(from = 12, to=20, by=2))
# Call the network topology analysis function
sft = pickSoftThreshold(Log2FC, powerVector = powers, verbose = 3)
# Plot the results:
par(mfrow = c(1,2))
cex1 = 0.9

# Scale-free topology fit index as a function of the soft-thresholdin
g power
plot(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2]
,
     xlab="Soft Threshold (power)",ylab="Scale Free Topology Model Fi
t,signed R^2",type="n",
     main = paste("Scale independence"))
text(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2]
,
     labels=powers,cex=cex1,col="red")

# this line corresponds to using an R^2 cut-off of h
abline(h=0.90,col="red")
# Mean connectivity as a function of the soft-thresholding power
plot(sft$fitIndices[,1], sft$fitIndices[,5],
     xlab="Soft Threshold (power)",ylab="Mean Connectivity", type="n"
,
     main = paste("Mean connectivity"))
text(sft$fitIndices[,1], sft$fitIndices[,5], labels=powers, cex=cex1,
col="red")
```

```
pickSoftThreshold: will use block size 1565.
 pickSoftThreshold: calculating connectivity for given powers...
   ..working on genes 1 through 1565 of 1565
   Power SFT.R.sq  slope truncated.R.sq mean.k. median.k. max.k.
1      1    0.396 -1.050          0.756 242.000  2.27e+02  455.0
2      2    0.916 -1.380          0.928  66.900  5.27e+01  220.0
3      3    0.975 -1.370          0.969  26.100  1.52e+01  137.0
4      4    0.973 -1.300          0.970  12.900  5.09e+00   97.9
5      5    0.970 -1.250          0.978   7.470  1.89e+00   75.3
6      6    0.961 -1.180          0.972   4.830  7.66e-01   60.6
7      7    0.965 -1.150          0.973   3.380  3.37e-01   50.8
8      8    0.949 -1.130          0.962   2.500  1.55e-01   43.9
9      9    0.970 -1.090          0.986   1.920  7.44e-02   38.6
10    10    0.974 -1.070          0.988   1.520  3.67e-02   34.2
11    12    0.956 -1.040          0.954   1.020  1.00e-02   27.7
12    14    0.951 -1.030          0.951   0.724  2.74e-03   22.9
13    16    0.962 -1.020          0.971   0.537  8.03e-04   19.3
14    18    0.964 -1.000          0.969   0.412  2.52e-04   16.5
15    20    0.966 -0.995          0.969   0.323  7.81e-05   14.2
```



In [0]:
```
%%R
#We choose the lowest power for which the scale-free topology fit ind
ex reaches 0.90
#calculate the adjacencies, using the soft thresholding power:
softPower = 2
adjacency = adjacency(Log2FC, power = softPower)
```

But this matrix give only information about the expression correlation between genes and the WGCNA methodology suggest that co-expression is not enough and the similarity between genes should be reflected at the network topology level.

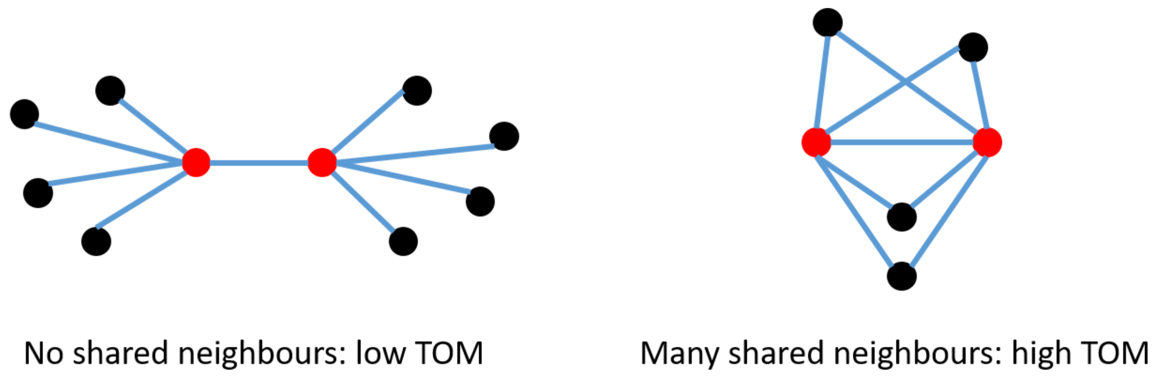
The **topological overlap matrix** $\Omega = [\omega_{ij}]$ measures direct connection + shared neighbours:

$$\omega_{ij} = \frac{l_{ij} + a_{ij}}{\min k_i, k_j + 1 - a_{ij}}$$

where $l_{ij} = \sum_u a_{iu}a_{uj}$ and $k_i = \sum_u a_{iu}$ is the node connectivity.

```
In [0]: Image('TOMmeasure.png',width=600)
```

Out[0]:

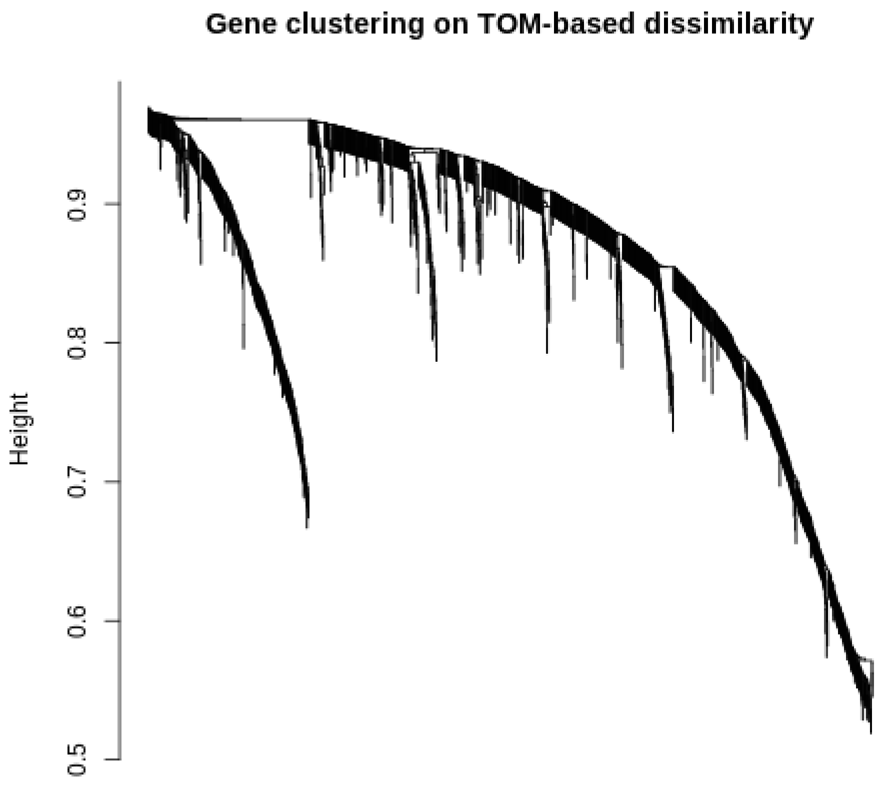

No shared neighbours: low TOM
Many shared neighbours: high TOM

```
In [0]: %%R
        #Topological Overlap Matrix (TOM)
        #To minimize effects of noise and spurious associations,
        #we transform the adjacency into Topological Overlap Matrix,
        #and calculate the corresponding dissimilarity:
        TOM = TOMsimilarity(adjacency)
        dissTOM = 1-TOM
```

```
..connectivity..
..matrix multiplication (system BLAS)..
..normalization..
..done.
```

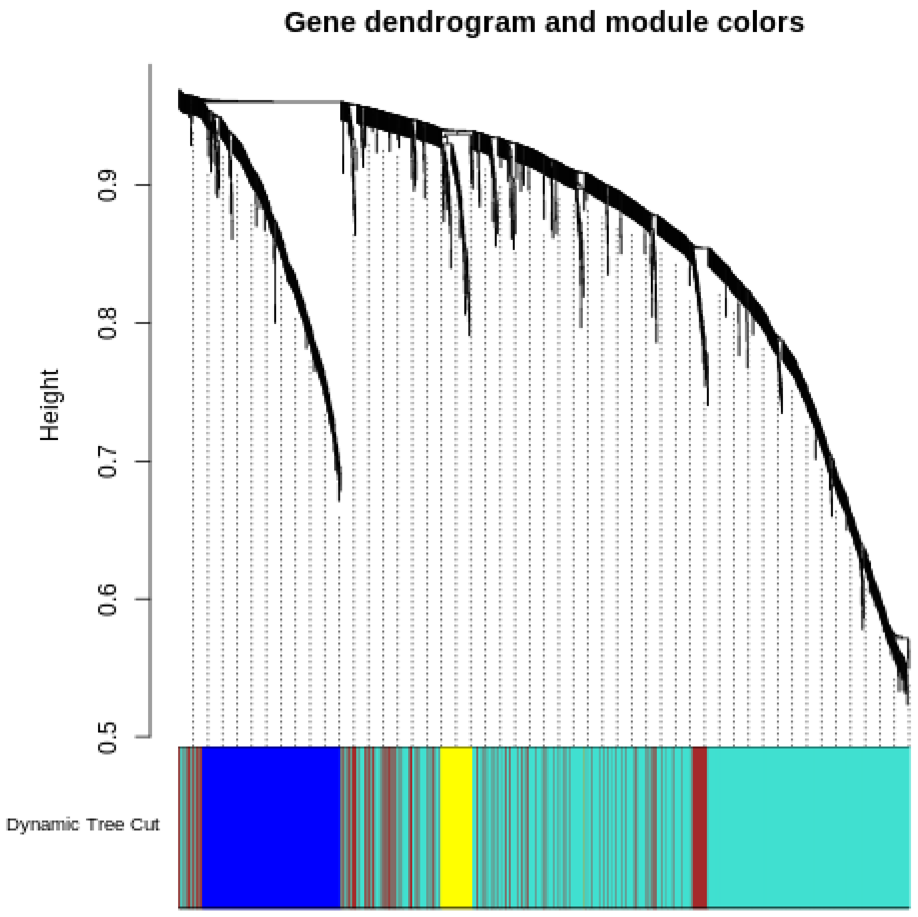
To group genes with coherent expression profiles into modules, WGCNA use **average linkage hierarchical clustering** coupled with the TOM-based dissimilarity.

In [0]:
```
%%R
#Clustering using TOM
# Call the hierarchical clustering function
geneTree = hclust(as.dist(dissTOM), method = "average")

# Plot the resulting clustering tree (dendrogram)
options(repr.plot.width=12, repr.plot.height=9)
plot(geneTree, xlab="", sub="", main = "Gene clustering on TOM-based
 dissimilarity",
     labels = FALSE, hang = 0.04)
```



In [0]:
```
%%R
# Module identification amounts to the identification of individual b
ranches
#('cutting the branches off the dendrogram')
# set the minimum module size relatively high in order to get large m
odules:
minModuleSize = 20;
# Module identification using dynamic tree cut:
dynamicMods = cutreeDynamic(dendro = geneTree, distM = dissTOM,
                            deepSplit = 2, pamRespectsDendro = FALSE,
                            minClusterSize = minModuleSize);
#show how many modules were identified and what the module sizes are
#The label 0 is reserved for genes outside of all modules.
table(dynamicMods)
# Convert numeric lables into colors
dynamicColors = labels2colors(dynamicMods)
table(dynamicColors)
# Plot the dendrogram and colors underneath
options(repr.plot.width=8, repr.plot.height=6)
plotDendroAndColors(geneTree, dynamicColors, "Dynamic Tree Cut",
                    dendroLabels = FALSE, hang = 0.03,
                    addGuide = TRUE, guideHang = 0.05,
                    main = "Gene dendrogram and module colors")
```

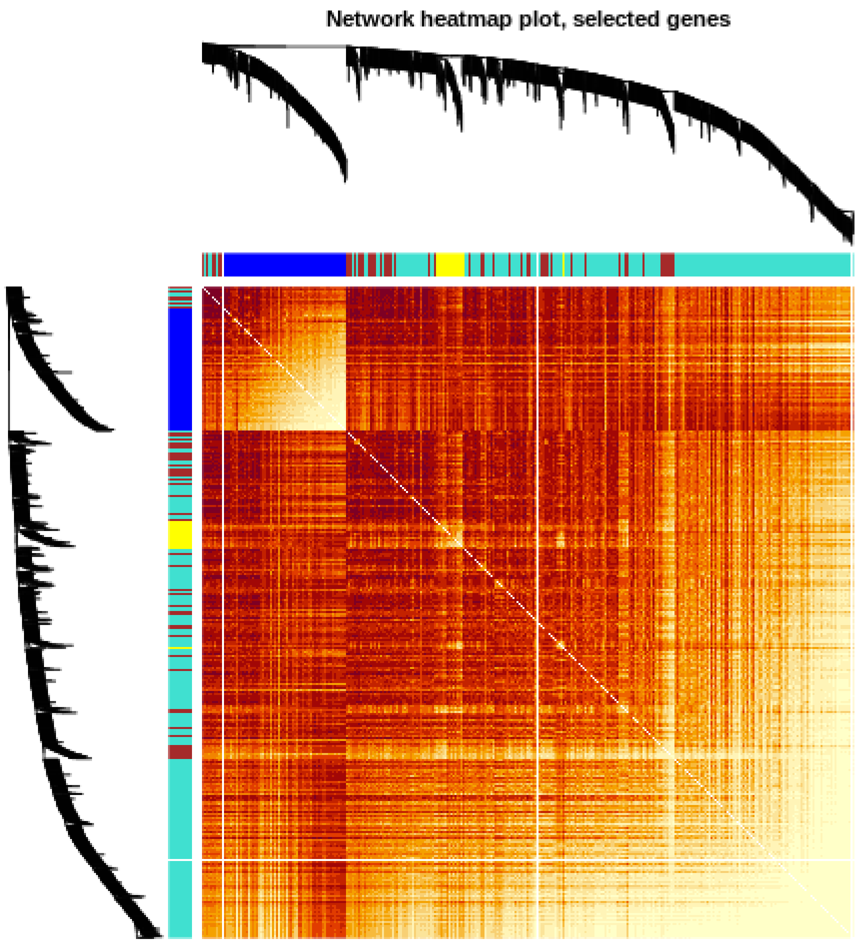
```
 ..cutHeight not given, setting it to 0.966  ===>  99% of the (trunca
ted) height range in dendro.
 ..done.
```



In [0]:
```
%%R
table(dynamicColors)
```

```
dynamicColors
     blue     brown turquoise    yellow 
      296       225       975        69 
```

In [0]:
```
%%R
# Taking the dissimilarity to a power makes the plot more informative
by effectively changing
# the color palette; setting the diagonal to NA also improves the cla
rity of the plot
plotDiss = dissTOM^10;
diag(plotDiss) = NA;
TOMplot(plotDiss, geneTree, dynamicColors, main = "Network heatmap pl
ot, selected genes")
```



In [0]:
```
%%R
# Recalculate MEs with color labels
MEs0 = moduleEigengenes(Log2FC, dynamicColors)$eigengenes
MEs = orderMEs(MEs0)

dim(MEs)
```

```
[1] 92  4
```

In [0]:
```
%%R
head(MEs)
```

```
                MEbrown MEturquoise       MEblue    MEyellow
GSM2596381  0.02134138 -0.02617441  0.002268229  0.15425525
GSM2596385  0.06847101 -0.09766816  0.111996636  0.08577636
GSM2596389 -0.10059016 -0.05732832  0.034750811  0.14208637
GSM2596393 -0.02872163  0.10661682  0.015534505 -0.09925217
GSM2596397 -0.27437467  0.10540385  0.068337959  0.15453848
GSM2596401 -0.02166658  0.17645468 -0.041811353 -0.06567029
```

Module Selection with LASSO

In [0]:
```
# Load phenotypic data
Na_K = pd.read_csv('Na_K_Shoot.csv', delimiter=' ', header=None)
```

LASSO (Least Absolute Shrinkage and Selection Operator) is a **regularized linear regression technique**, a method that combines a regression model with a procedure of contraction of some parameters towards zero and selection of variables, imposing a restriction or a penalty on the regression coefficients.

Very usefull in problems where the number of variables (genes) n is much greater than the number of samples p ($n \gg p$)

Lasso solves the least squares problem with restriction on the L_1-norm of the coefficient vector minimizing:

$$\sum_{i=1}^{p}\left(y_i - \sum_{j=1}^{n}\beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{n}|\beta_j|$$

being s, $\lambda \geq 0$ the respective penalty parameters for complexity.

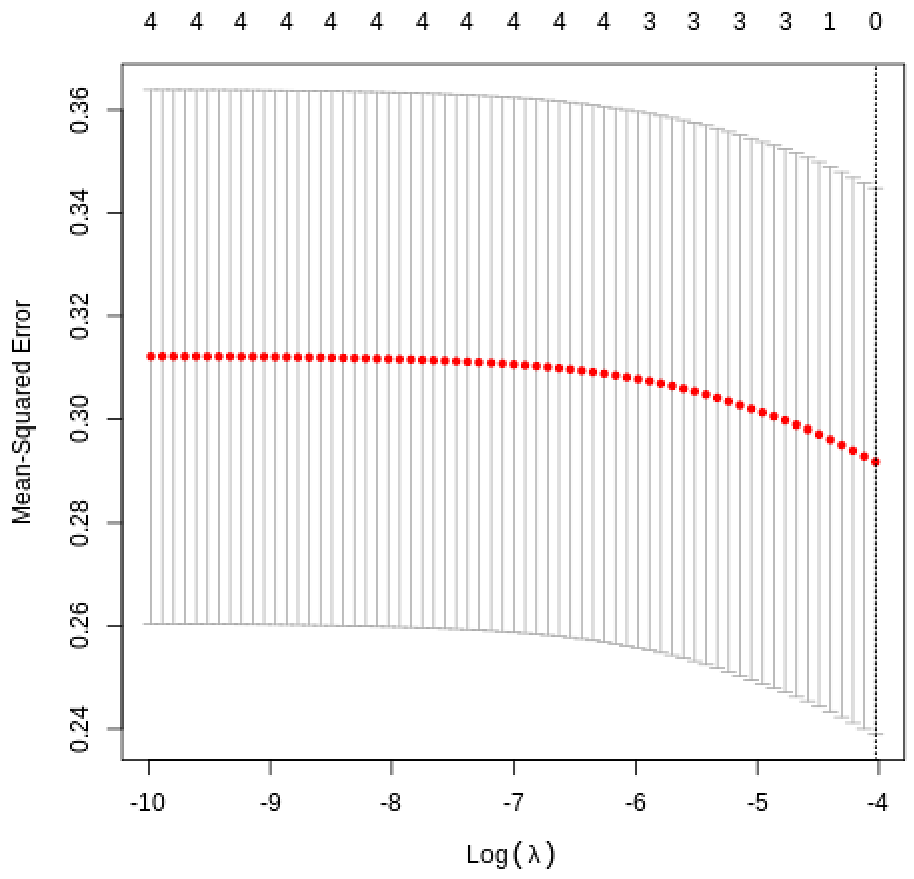
In [0]:
```
%%R -i Na_K

# Remove NAs
Data <- cbind(Na_K,MEs)
Data <- na.omit(Data)

# Name of the dependent variable
names(Data)[1]= 'trait'

# Inspect the data
head(Data)
```

```
     trait     MEbrown MEturquoise       MEblue    MEyellow
0 7.192423  0.02134138 -0.02617441  0.002268229  0.15425525
1 7.344413  0.06847101 -0.09766816  0.111996636  0.08577636
2 7.647553 -0.10059016 -0.05732832  0.034750811  0.14208637
3 7.368621 -0.02872163  0.10661682  0.015534505 -0.09925217
4 7.989185 -0.27437467  0.10540385  0.068337959  0.15453848
5 7.944348 -0.02166658  0.17645468 -0.041811353 -0.06567029
```

In [0]:
```
%%R
# Additionnal data preparation
x <- model.matrix(trait~., Data)[,-1]
y<- Data$trait
cat('dim(x)= ',dim(x),'\n')
cat('dim(y)= ',length(y))
```

```
dim(x)=  92 4
dim(y)=  92
```

In [0]:
```
%%R
# Fit the lasso penalized regression model:
# Find the best lambda using cross-validation
cv.lasso <- cv.glmnet(x, y, alpha = 1)
plot(cv.lasso)
```



In [0]:
```
%%R
# The lambda that minimizes the prediction error
#This lambda value will give the most accurate model
print(cv.lasso$lambda.min)
```

```
[1] 0.01782704
```

In [0]:
```
%%R
# Fit the model
model <- glmnet(x, y, alpha = 1, lambda = cv.lasso$lambda.min)
# Display regression coefficients
cf.bestlambda <-coef(model)
df3 <- as.data.frame(summary(cf.bestlambda))
df3$Gene <- rownames(cf.bestlambda)[df3$i]
df3$Beta <- colnames(cf.bestlambda)[df3$j]
dim(df3)[1]
df3[c(3,4)]
```

```
         x        Gene
1 7.545888 (Intercept)
2 0.000000     MEbrown
```

In [0]:
```
%%R -o Module
color <-'brown'  # replace the module color
Module <- Log2FC[,dynamicColors==color]
print(dim(Module))
```

```
[1]  92 225
```

In [0]:
```
Module.columns
```

Out[0]:
```
Index(['X13102.t00559', 'X13106.t03437', 'X13104.t02991', 'X13108.t03
707',
       'X13109.t03178', 'X13101.t00929', 'X13101.t00476', 'X13108.t02
563',
       'X13104.t04535', 'X13110.t03261',
       ...
       'X13104.t04954', 'X13111.t03656', 'X13103.t01628', 'X13106.t03
720',
       'X13101.t03281', 'X13104.t03941', 'X13103.t00401', 'X13109.t03
047',
       'X13108.t03180', 'X13111.t03260'],
      dtype='object', length=225)
```