CS-E5885
Modeling biological networks
Spring 2024

Eeli Friman
903628

Table of contents

1 Introduction

Biological network inference methods are methods that are used to build a network model of a biological process using experimental data. Reverse engineering the network is important for understanding the dynamics and interactions of the underlying process. [1] The aim of this project was to apply network inference method(s) an experimental dataset, to build a gene regulatory network of yeast.

The synthetically constructed gene regulatory network contains five transcription factor (TF) genes: Swi5, Cbf1, Gal4, Gal80 and Ash1. The five genes are assumed to work in isolation from any other genes. The gene interactions are visualized in figure 1. The role of galactose in the networks is to act as a trigger to the transcription of the genes. [2]

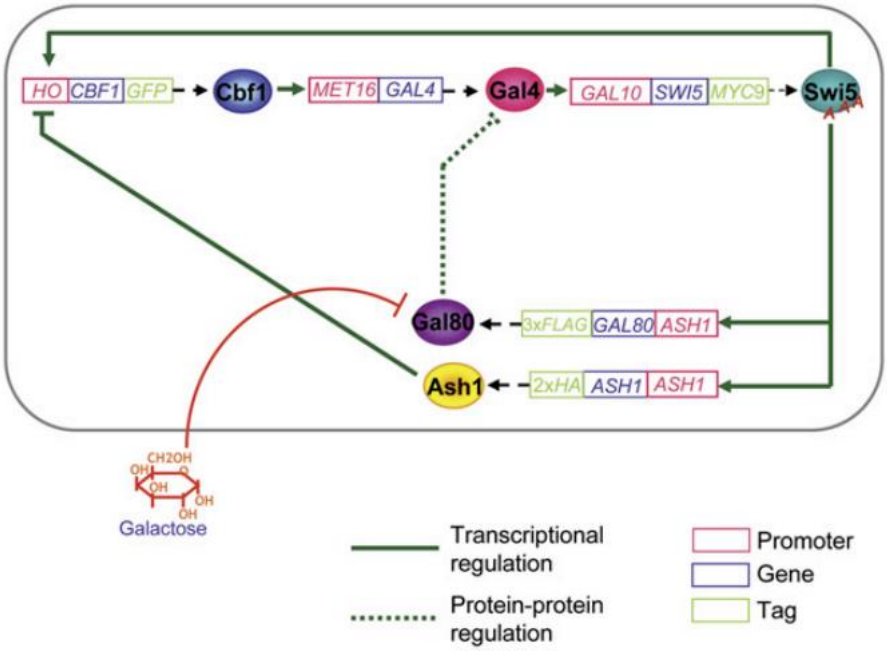

Figure 1. Synthetically constructed transcriptional network in yeast. [2]

In the scope of this project, a network is considered a graphical representation of a model, which consists of nodes and edges. Nodes of the graph represent variables and edges the interactions. [3] Due to the type of the methods used in this project, the graphical representations are undirected.

This report is divided into three sections. The second section introduces the dataset and the network inference methods. The final section analyzes the performance of the methods, derives the results, sources of error and final conclusions. The code used to obtain the results is included in the appendix.

2 Methods

This s section introduces the dataset and dives into the methods used in this project to construct the underlying biological process. The model choice for this project was relevance networks, which is an information-theoretic approach [2]. Relevance networks are methods that can be used to pairwise visualize and detect the interactions of variables [3]. Only the interactions that are stronger than a chosen threshold are chosen for the network. Strength of the interaction can be measured with either correlation or mutual information. [2]

Relevance networks were chosen for this project since they are a popular method of visualizing interactions between genes [3]. A relevance network can be built solely from the data, while assuming zero prior information. The networks can be altered by tuning the threshold, which acts as a detection rate. By varying the threshold, multiple models can be obtained, which allows for easy comparison between the real network and the proposed ones. The choice for the threshold proposes a balancing act between precision and detectability of the network.

2.1 Dataset

The dataset was obtained from a time-series experiment. In the experiment, the expressions of the different genes were measured in 10-minute time intervals. The experiment was conducted as a switch-off experiment, where the abundance of galactose was regulated at the beginning of the experiment. [2]

The time interval was split into 20 elements, ranging from 0 to 190 minutes. Each of the five TF genes had 20 expression measurements. There were no missing values in the dataset.

2.2 ARACNe algorithm

The mutual information relevance network was built using the ARACNe algorithm (Algorithm for the Reconstruction of Accurate Cellular Networks). ARACNe is a method which can be applied to a dataset to construct a two-way network. The algorithm analyzes the mutual information of two variables to find statistically dependent pairs and filters out redundant connections. [1] The method was chosen due to personal interests, as well as its applications in similar settings in literature.

The mutual information network is constructed by drawing an edge between variables x_i and x_j, if their mutual information, denoted as $I(X_i|X_j)$, is above a chosen threshold I_0. Mutual information is a measure of statistical dependency, that can be interpreted as how much variable X_j contains information of variable X_i [1, 4]. Information content of a variable can be defined as

$$I(X = x) = -\log p(X = x), \tag{1}$$

for a probability distribution function (PDF) $p(X = x)$. Average uncertainty of a random variable X is the expected information content, denoted as entropy $H(X)$. Entropy is defined as

$$H(X) = E[I(X)] = -\sum_{i=1}^{n} p(x_i) \log p(x_i), \quad (2)$$

where the sum is calculated over the n-elements of the distribution $p(x_i)$. [4]

The relative entropy of two distributions, $p(X)$ and $q(X)$, is a distance measure between the distributions. Relative entropy is also known as the Kullback-Leibler divergence D_{KL}, defined as

$$D_{KL}(p||q) = \sum_{i=1}^{n} p(x_i) \log \frac{p(x_i)}{q(x_i)}, \quad (3)$$

which is an asymmetric quantity, since $\frac{p(x_i)}{q(x_i)} \neq \frac{q(x_i)}{p(x_i)}$, assuming $p(x_i) \neq q(x_i)$. [4]

Mutual information of X_i and X_j, $I(X_i|X_j)$, is defined as the relative entropy of the joint distribution $p(X_i, X_j)$ and the marginal distributions $p(X_i)$ and $p(X_j)$. Mutual information obtained with the Kullback-Leibler divergence is

$$I(X_i|X_j) = \sum_{X_i, X_j} p(X_i, X_j) \log \frac{p(X_i, X_j)}{p(X_i)p(X_j)}, \quad (4)$$

where the summation is calculated over the values of the distributions of X_i and X_j. Mutual information is a symmetric quantity. [4]

The underlying probability distributions of the transcription factor genes are unknown. The distributions can be estimated using the Gaussian kernel density estimation (KDE), which is defined as

$$\hat{p}(\boldsymbol{x}|D) = \frac{1}{N} \sum_{i=1}^{N} Norm(\boldsymbol{x}|\boldsymbol{x}_i, \sigma^2 \boldsymbol{I}), \quad (5)$$

where N denotes the amount of observations of the estimated PDF. Here the $Norm$ abbreviates the normal distribution, $\boldsymbol{I}$ the $d\ x\ d$ identity matrix and σ^2 the tuning parameter, known as the bandwidth of the estimator. [3] The Gaussian kernel density estimate generates a Gaussian distribution around every datapoint. The total distribution is obtained by summing over the separate Gaussian distributions.

The distribution estimations were calculated using the tool KernelDensity from the scikit-learn package. The estimation was done with the Gaussian kernel, and the "auto" algorithm. The tool KernelDensity computes the uni- or bivariate Gaussian kernel density estimate, according to equation X, on given variable(s). The estimate is then fitted on a range around the PDF, using .fit. The logarithmic PDF is obtained from the estimate with score.samples. [5] Example of the KernelDensity applied on the gene expression data of Swi5 and Cbf1 are visualized in figures 2 and 3.

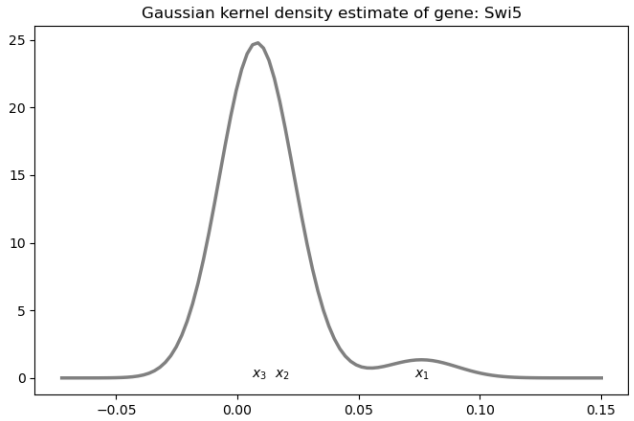

Figure 2: Gaussian marginal KDE of gene Swi5. The points x_i represent the elements of the dataset where the Gaussian distributions are drawn.

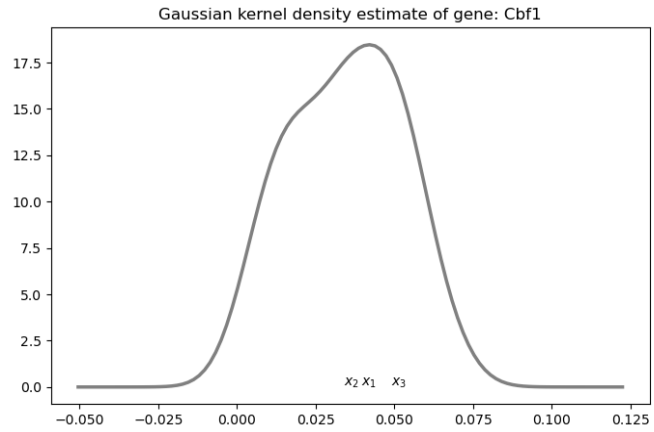

Figure 3. Gaussian marginal KDE of gene Cbf1. The points x_i represent the elements of the dataset where the Gaussian distributions are drawn.

The value of the bandwidth of the kernel density estimate, σ^2, was optimized separately for every marginal and joint distribution. The optimization was done with leave-one-out cross-validation (LOOCV). In LOOCV, the dataset of N samples is split into N folds. The score of each test set is evaluated and averaged across all the different combinations of folds for every threshold. LOOCV was performed with tools GridSearchCV and LeaveOneOut from the skicit-learn package. The tool GridSearchCV was used to find the best parameter value for the KernelDensity estimator. Total of 100 values of σ^2 were applied, ranging from 0.0001 to 10. The score of the parameter was evaluated with LeaveOneOut, which performs the LOOCV calculation on KernelDensity estimate with the test parameter value. [5]

The ARACNe algorithm uses the kernel density probability distribution estimates to calculate the mutual information $I(X_i|X_j)$ for all the gene combinations [1]. The mutual information can be calculated using the KDEs of marginal distributions $\hat{p}(X_i)$ and $\hat{p}(X_j)$ and joint distributions $\hat{p}(X_i, X_j)$ as

$$\hat{I}(X_i|X_j) = \frac{1}{N_{X_i}}\frac{1}{N_{X_j}}\sum_{N_{X_i},N_{X_j}} \hat{p}(X_i, X_j|D) \log\frac{\hat{p}(X_i,X_j|D)}{\hat{p}(X_i|D)\hat{p}(X_j|D)} \quad (7)$$

where $N = N_{X_i} = N_{X_j}$ denote the intervals for the PDFs $\hat{p}(X_i|D)$ and $\hat{p}(X_j|D)$. Mutual information was calculated for every pair of X_i, X_j, where $i \neq j$. A 5-by-5 mutual information matrix M was constructed, where

$$M_{ij} = \hat{I}\left(X_i\middle|X_j\right) \quad (8)$$

Due to symmetricity of mutual information, only the upper diagonal of the matrix M was calculated for time-saving reasons. On the diagonal, where $i = j$, the mutual information is 0, since from equation 3, $\log\frac{p(x_i)}{q(x_i)} = 0$, for $q(x_i) = p(x_i)$, The values of the mutual information matrix were normalized to be in range 0 and 1.

In ARACNe algorithm, the final network is constructed by examining all the triplets of variables x_i, x_j and x_k, which have a mutual information value above the threshold I_0. The edge between the pair with the smallest mutual information value is removed from the network. [1]

2.3 Correlation

The second analyzing method chosen was correlation. Correlation measure can be used to detect interactions between variables. The correlation measure used in this project is the Pearson correlation, which measure the linear correlation between two variables. Pearson correlation was chosen, since it is easy and fast to implement, and computationally inexpensive to apply. [3]

The correlation coefficient is a value between -1 and 1. Correlation value of 0 indicates no linear correlation. Absolute values close to 1 indicate strong correlation between variables. Different correlation coefficient values are visualized in figure 4.

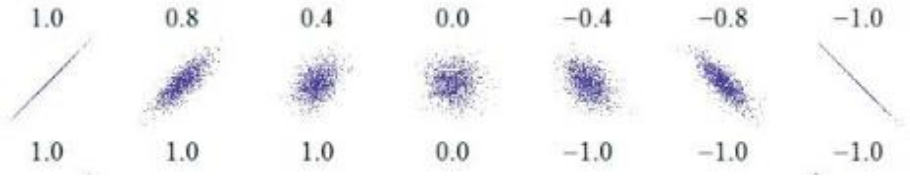

Figure 4. Pearson correlation between two variables. [3]

Pearson correlation applied on variables X and Y of a population is defined as

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}, \tag{9}$$

where σ is the denotes the standard deviation, and $cov(X,Y)$ is covariance. [3, 6] Covariance of a population is defined as

$$cov(X,Y) = E[(X - \mu_X)(Y - \mu_Y)], \tag{10}$$

where μ denotes the mean of a variable [3]. The Pearson correlation of a sample is defined as

$$r = \frac{\sum(X - m_X)(Y - m_Y)}{\sqrt{\sum(X - m_X)^2 \sum(Y - m_Y)^2}}, \tag{11}$$

where m_X and m_Y are the respective sample means of variables X and Y. [7]

The empirical correlation matrix was calculated using a correlation coefficient tool, pearsonr from the SciPy stats module. The pearsonr tool calculates the Pearson correlation matrix and the associated p-values for a set of variables. [7] The Pearson correlation is calculated as in equation 11. The correlation matrix is constructed as

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} C_{jj}}}, \tag{12}$$

where C denotes covariance, and R_{ij} is the Pearson sample correlation between variables x_i and x_j. For diagonal elements, the value of the empirical correlation matrix is 1. [3, 6]

The correlation network is obtained by determining the relevant variables according to the correlation measure. Selection criteria is determined with a threshold τ, which is a value between values 0 and 1. In the network, an edge is drawn between nodes x_i and x_j, if the correlation matrix value corresponding to the variables is larger than or equal to the chosen threshold, i.e.

$$|R_{ij}| \geq \tau \tag{13}$$

Since the correlation matrix is symmetric, only the upper diagonal values are calculated.

The downside of using Pearson correlation measure is that it only measures the linear correlation between values. Pearson correlation is therefore unable to detect higher dimensional dependence between variables. Correlation is also highly sensitive to outliers. [8]

3 Results

This section analyzes the observations and proposed models obtained by applying the background theory from section 2. The two methods with differing thresholds are compared. The models are analyzed to determine the true and false positive rates proposed gene interactions of the models.

3.1 Model performance

Both ARACNe algorithm and correlation were applied with varying threshold values I_0 and τ. Multiple thresholds proposed multiple network models with different detection rates. For both models, the amount of total proposed interactions were calculated. The fraction of true and false positive gene interactions were obtained by comparing the proposed total interactions to that of the model. In the original model, there are a total of 6 gene-gene interactions present among the 5 TF-genes [2]. The interactions are summarized in table 1.

Table 1. True gene-gene interactions. [2]

TF Gene	Swi5	Cbf1	Gal4	Gal80	Ash1
Interacts with (transcriptional regulation)	Cbf1 Gal80 Ash1	Gal4	Swi5		Cbf1

In ARACNe algorithm, the value of threshold I_0 was varied between 0.2 and 0.9. The kernel density estimates were all fit for the variables on a range from $\min(x_i) - 0.02$ to $\max(x_i) - 0.02$. The marginal and joint PDFs were all estimated with the optimal bandwidth obtained with LOOCV. Mutual information of every gene pair was calculated, and a matrix was constructed according to equation 8. The mutual information matrix is visualized in figure 5.

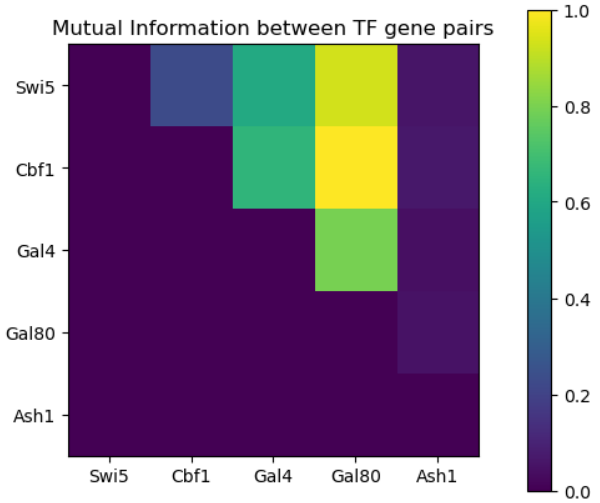

Figure 5. Mutual information between TF gene pairs. The values are normalized to be in range 0 and 1.

After obtaining the mutual information matrix, the values below the threshold I_0 were discarded. The process was repeated for all values of I_0 in range 0.2 – 0.9. The mutual information matrix obtained after filtering with threshold $I_0 = 0.4$ is visualized in figure 6.

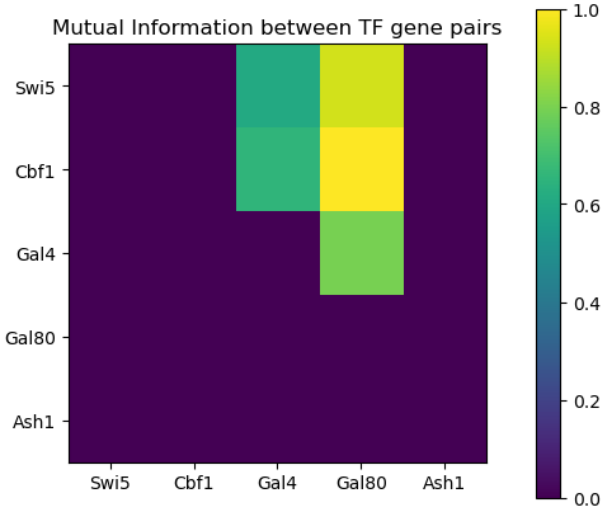

Figure 6. Mutual information between TF gene pairs with threshold $I_0 = 0.4$. The values are normalized to be in range 0 and 1.

Finally, all the triplets of the remaining mutual information values were compared, and the smallest values of the triplets were discarded. The final matrix obtained through ARACNe, with $I_0 = 0.4$, is visualized in Figure 7.

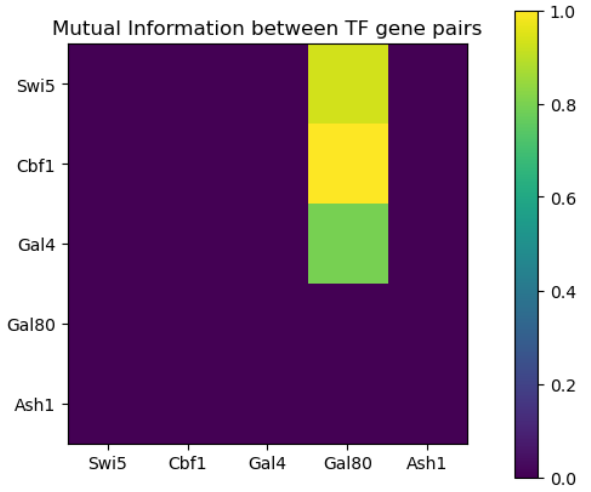

Figure 7. Mutual information between TF gene pairs with threshold $I_0 = 0.4$. The values are normalized to be in range 0 and 1. The least MI values of triplets of genes are filtered out.

Table 2 summarizes the performance of all the different models through total, true positive and false positive, true negative and false negative interactions. A ROC curve representation of the model performances is visualized in figure 8. In the ROC plot, the true positive rate is plotted against the false positive rate.

Table 2. Total, true positive and false positive, true negative and false negative interactions.

Model (I0)	Total interactions	True positive	False positive	True negative	False negative
0.2	3	1	2	2	5
0.3	3	1	2	2	5
0.4	3	1	2	2	5
0.5	3	1	2	2	5
0.6	3	1	2	2	5
0.7	3	1	2	2	5
0.8	2	1	1	3	5
0.9	2	1	1	3	5

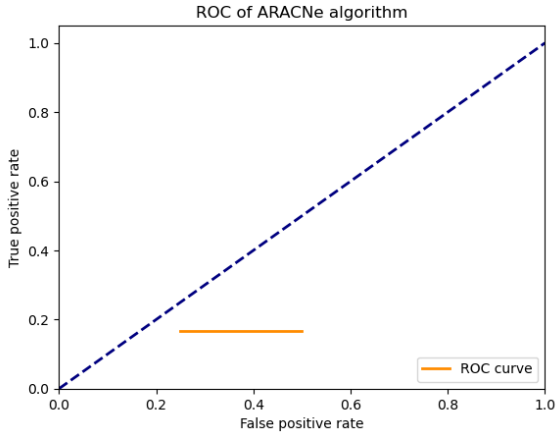

Figure 8. ROC curve of the mutual information networks with varying thresholds.

In correlation network analysis, the threshold τ was varied between 0.1 and 0.9. The correlation matrix was calculated with equation 13. The Pearson correlation matrix is visualized in figure 9.

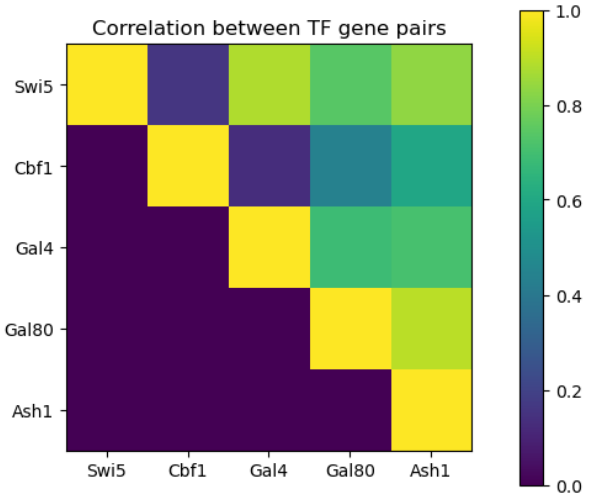

Figure 9. Correlation between TF gene pairs.

For each model with varying threshold, the correlation matrix entries R_{ij} below the threshold were discarded. The correlation matrix obtained after filtering with threshold $\tau = 0.6$ is visualized in figure 10.

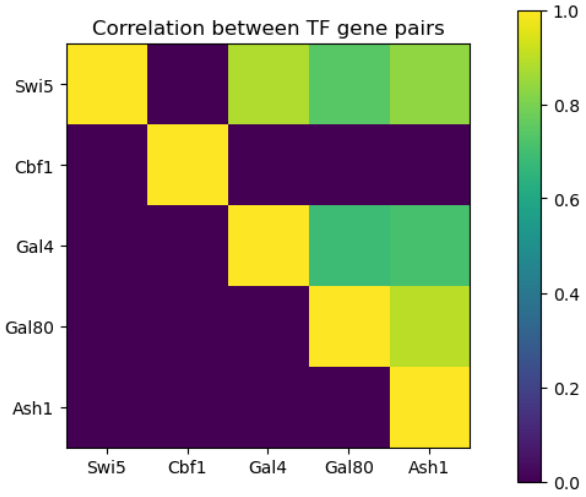

Figure 10. Correlation between TF gene pairs with threshold $\tau = 0.6$.

The performance of the correlation models are summarized in table 3. A ROC curve representation of the model performances is visualized in figure 11.

Table 3. Total, true positive and false positive, true negative and false negative interactions of the correlation network.

Model (τ)	Total interactions	True positive	False positive	True negative	False negative
0.1	10	6	4	0	0
0.2	8	4	4	0	2
0.3	8	4	4	0	2
0.4	8	4	4	0	2
0.5	7	4	3	1	2
0.6	6	3	3	1	3
0.7	5	3	2	2	3
0.8	3	2	1	3	4
0.9	0	0	0	4	6

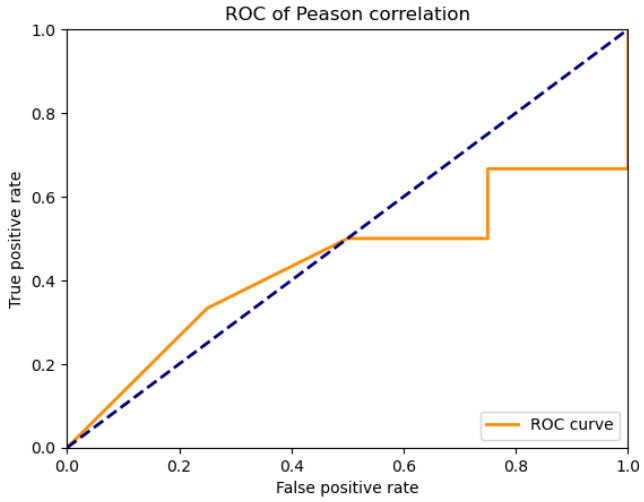

Figure 11. ROC curve of the correlation networks with varying thresholds.

3.2 Error analysis

The validity and performance of the models may have been affected by multiple factors. A common error causing factor in relevance network analysis are spurious correlations. Spurious correlations refer to correlations among variables, that are due to other variables rather than direct interactions. Some of the correlations may be due to other regulating variables, which are reflected as a false correlation or mutual information value.

Pearson correlation measures only the linear correlation between variables. Therefore, it is unable to detect any higher dimensional interactions. The undetected true interactions were most likely missed due to the linearity assumption of the correlation model. For example, the true TF-gene interaction between genes Swi5 and Cbf1 had a Pearson correlation value of 0.15.

Error in ARACNe may have also been due to the implementation of the algorithm. Getting the algorithm to work properly turned out troublesome, which may have caused faulty mutual information values. Some of the kernel density estimates may have also been incorrect, due to potential outliers in the dataset. Every first measurement of gene expression was relatively large compared to every other value, which might have caused the faulty estimates of the PDFs.

3.3 Conclusion

The ROC plot of ARACNe network (figure 8) shows that the method performed poorly. The detection rate of true positives remained constant across the models, while the amount of false positives increased. The ROC plot of correlation network (figure 11) shows that the models with thresholds around 0.5 had a decent true positive-false positive ratio.

The best model obtained with ARACNe predicted only 1 of the 6 TF-interactions correctly, with 3 true negatives and a single false positive. The final step of ARACNe, analyzing the triplets, discarded almost every true interaction. This means that the mutual information was larger with false interactions than the true interactions.

The best correlation model, $\tau = 0.7$, predicted 3 out of the 6 TF-interactions correctly. However, the model predicted 2 false positives and only 2 true negatives. Still, the Pearson correlation performed better than ARACNe.

It is clear from the ROC, that the ARACNe algorithm did not prosper in constructing the regulation network. For better results, a different implementation of kernel density estimates could produce better results, w.r.t. section 3.2, error analysis. Switching Pearson correlation to a higher dimensional measure could also be beneficial.

References

1. Margolin, A.A., et al., ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context. BMC Bioinformatics, 2006. 7(1): p. S7.
2. Irene, C., et al., A Yeast Synthetic Network for In Vivo Assessment of Reverse-Engineering and Modeling Approaches. Cell, 2009. 137(1): p. 172-181.
3. Murphy, K.P., Machine learning : a probabilistic perspective. Adaptive computation and machine learning series. 2012, London, England: MIT Press.
4. Cover, T.M. and J.A. Thomas, Elements of information theory. 2nd ed. 2006, Hoboken, N.J: Wiley-Interscience.
5. Pedregosa, F., et al., Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2011. 12: p. 2825--2830.
6. Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. 2020.
7. Virtanen, P., et al., & SciPy 1.0 Contributors (2020), SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 2020. 17: p. 261-272.
8. Watters, P. and S. Boslaugh, Statistics in a Nutshell. 1st ed. 2008: O'Reilly Media Inc.

Appendix

PROJECT

March 4, 2024

1 Assignment project

CS-E5885 Modeling Biological Networks

Student: Eeli Friman Student number: 903628

[ ]:
```
import sys
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
from sklearn.neighbors import KernelDensity
import random
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut
from scipy import stats
import matplotlib.colors as mcolors
%cd /home/frimane1/notebooks/CS-E5885/Project
```

[ ]:
```
data = open("data.txt", "r")
next(data)

time = []
SWI5 = []
CBF1 = []
GAL4 = []
GAL80 = []
ASH1 = []

for line in data:
    values = line.strip().split()
    time.append(float(values[0]))
    SWI5.append(float(values[1]))
    CBF1.append(float(values[2]))
    GAL4.append(float(values[3]))
    GAL80.append(float(values[4]))
    ASH1.append(float(values[5]))

time = np.array(time)
```

```
SWI5 = np.array(SWI5)
CBF1 = np.array(CBF1)
GAL4 = np.array(GAL4)
GAL80 = np.array(GAL80)
ASH1 = np.array(ASH1)

vars = np.column_stack((SWI5, CBF1, GAL4, GAL80, ASH1))
```

[ ]:
```
plt.figure(figsize=(10, 6))
for i in range(len(vars[0])):
    plt.plot(time, vars[:, i], label=f'Variable {i+1}')

plt.xlabel('Time (t)')
plt.ylabel('Arbitrary Units')
plt.title('Time-course Data')
plt.legend()
plt.grid(True)
plt.show()
```

METHOD 1 CORRELATION NETWORK

[ ]:
```
Pcorr = np.identity(5)
pvalue = np.identity(5)
n_rows, n_cols = Pcorr.shape
for i in range(n_rows):
    for j in range(i + 1, n_cols):
        Pcorr[i, j] = stats.pearsonr(vars[:, i], vars[:, j])[0]
        pvalue[i, j] = stats.pearsonr(vars[:, i], vars[:, j])[1]
```

[ ]:
```
def plot_corr(corr):
    plt.figure(figsize=(5,5))

    plt.imshow(corr,interpolation='None')
    plt.colorbar(fraction=0.05, pad=0.1)
    plt.xticks([0, 1, 2, 3, 4], ['Swi5', 'Cbf1', 'Gal4', 'Gal80', 'Ash1'])
    plt.yticks([0, 1, 2, 3, 4], ['Swi5', 'Cbf1', 'Gal4', 'Gal80', 'Ash1'])

    plt.title('Correlation between TF gene pairs');
```

[ ]:
```
def filter_corr(corr, threshold):
    filtered_corr = np.copy(corr)
    for i in range(len(corr)):
        for j in range(i + 1, len(corr[i])):
            if filtered_corr[i, j] <= threshold:
                filtered_corr[i, j] = 0
```

```
    return filtered_corr
```

[ ]:
```
plot_corr(Pcorr)
Networks = []
Threshold = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
for k in range(len(Threshold)):
    Networks.append(filter_corr(Pcorr, Threshold[k]))


plot_corr(Networks[4])
```

METHOD 2

MUTUAL INFORMATION

[ ]:
```
def marginal_kde(variable, N, bw):
    xrange = var_range(variable, N)
    data = dataset2D(variable)

    kde = KernelDensity(kernel = 'gaussian', bandwidth = bw, algorithm =␣
 ↪'auto').fit(data)
    logP = kde.score_samples(xrange)

    return logP
```

[ ]:
```
def dataset2D(x):
    D = x[:, np.newaxis]

    return D
```

[ ]:
```
def var_range(variable, N):
    d = variable.max() - variable.min()
    var_range =  np.linspace(variable.min()-d, variable.max()+d, num=N)
    var_range = var_range[:, np.newaxis]

    return var_range
```

[ ]:
```
def joint_kde(x, y, N, bw):

    Pxy_range = combined_range(x, y, N)
    data_combined = joint_dataset(x, y)


    kde = KernelDensity(kernel = 'gaussian', bandwidth = bw).fit(data_combined)
    logPxy = kde.score_samples(Pxy_range)

    return logPxy
```

[ ]:
```
def joint_dataset(x1, x2):
    xi = np.full(20, x1)
    x1x2 = np.stack((xi, x2)).T

    return x1x2
```

[ ]:
```
def combined_range(x, y, N):
    xi_range =  np.linspace(x-0.02, x+0.02, num=N)
    y_range = np.linspace(y.min()-0.02, y.max()+0.02, num=N)

    return np.stack((xi_range, y_range)).T
```

[ ]:
```
def opt_bw_marginal(x):
    bandwidths = 10 ** np.linspace(-4, 1, 100)

    data = dataset2D(x)

    grid = GridSearchCV(KernelDensity(kernel='gaussian'), {'bandwidth':␣
 ↪bandwidths}, cv=LeaveOneOut())
    grid.fit(data);
    best_bw = grid.best_params_

    return best_bw.get('bandwidth')
```

[ ]:
```
def opt_bw_joint(x, y):
    bandwidths = 10 ** np.linspace(-4, 1, 100)

    data = joint_dataset(x, y)

    grid = GridSearchCV(KernelDensity(kernel='gaussian'), {'bandwidth':␣
 ↪bandwidths}, cv=LeaveOneOut())
    grid.fit(data);
    best_bw = grid.best_params_

    return best_bw.get('bandwidth')
```

[ ]:
```
def plot_KDEs(vars, N):
    Genes = ['Swi5', 'Cbf1', 'Gal4', 'Gal80', 'Ash1']
    for i in range(len(vars[1])):
        plt.figure(figsize=(8, 5))
        x = vars[:, i]
        marg = marginal_kde(x, N, opt_bw_marginal(x))

        plt.plot(var_range(x, N), np.exp(marg), color='gray', linewidth=2.5)
        plt.title('Gaussian kernel density estimate of gene: {}'.
 ↪format(Genes[i]))
```

```
        for k in range(3):
            xi = x[k]
            plt.annotate(r'$x_{}$'.format(k+1), xy=[xi, 0.
 ↪05],horizontalalignment='center', fontsize=10)
```

[ ]:
```
plot_KDEs(vars, 100)
```

[ ]:
```
MI = np.zeros((5, 5))
nrows = MI.shape[0]
ncols = MI.shape[1]
N = 100
sum = 0
for i in range(nrows): # loop for upper diagonal
    marginal_xk = marginal_kde(vars[:,i], N, opt_bw_marginal(vars[:,i]))

    for j in range(i + 1, ncols):
        MI_val = 0
        marginal_xl = marginal_kde(vars[:,j], N, opt_bw_marginal(vars[:,j]))

        for c in range(20):
            kde_xkxl = joint_kde(vars[c,i], vars[:,j], N,␣
 ↪opt_bw_joint(vars[c,i], vars[:,j]))

            for k in range(N):
                for l in range(N):
                    MI_val += np.exp(kde_xkxl[l]) * (kde_xkxl[l] -␣
 ↪marginal_xk[k] + marginal_xl[l])
                    sum+=1


        MI[i,j] = (1/(N**2)) * MI_val
        print("success")
```

[ ]:
```
upper_diag_values = []
for i in range(len(MI)):
    for j in range(i + 1, len(MI[i])):
        upper_diag_values.append(MI[i][j])

max_value = max(upper_diag_values)

new_min = 0
new_max = 1

scaled_MI = []
```

```
for i in range(len(MI)):
    scaled_row = []
    for j in range(len(MI[i])):
        if j <= i:
            scaled_row.append(MI[i][j])
        else:
            scaled_value = (max(MI[i][j], 0) / max_value) * (new_max - new_min)␣
 ↪+ new_min
            scaled_row.append(scaled_value)
    scaled_MI.append(scaled_row)

for row in scaled_MI:
    for value in row:
        print("{:.12f}".format(value), end=" ")
    print()
```

[ ]:
```
def plot_MI(MI):
    plt.figure(figsize=(5,5))

    plt.imshow(MI,interpolation='None')
    plt.colorbar(fraction=0.05, pad=0.1)
    plt.xticks([0, 1, 2, 3, 4], ['Swi5', 'Cbf1', 'Gal4', 'Gal80', 'Ash1'])
    plt.yticks([0, 1, 2, 3, 4], ['Swi5', 'Cbf1', 'Gal4', 'Gal80', 'Ash1'])

    plt.title('Mutual Information between TF gene pairs');
```

[ ]:
```
def filter_MI(MI, threshold):
    filtered_MI = np.copy(MI)
    for i in range(len(MI)):
        for j in range(i + 1, len(MI[i])):
            if filtered_MI[i, j] <= threshold:
                filtered_MI[i, j] = 0

    return filtered_MI
```

[ ]:
```
def filter_smallest(MI):

    least_of_three = []

    for i in range(MI.shape[0]):
        idx_j = np.where(MI[i] != 0)[0]
        for j in idx_j:
            idx_k = np.where(MI[j] != 0)[0]
            for k in idx_k:

                if MI[i,k] != 0:
```

```
                    values = [MI[i][j], MI[i][k], MI[j][k]]
                    min_value_index = values.index(min(values))

                    if min_value_index == 0:
                        least_of_three.append((i, j))
                    elif min_value_index == 1:
                        least_of_three.append((i, k))
                    else:
                        least_of_three.append((j, k))



    for i in least_of_three:
        MI[i] = 0

    return MI
```

[ ]:
```
Thresholds_I0 = [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
scaled_MI = np.array(scaled_MI)
Networks_MI = []
for i in range(len(Thresholds_I0)):
    temp2 = np.copy(scaled_MI)
    temp2 = filter_MI(temp2, Thresholds_I0[i])
    Networks_MI.append(temp2)

for i in range(len(Networks_MI)):
    temp3 = np.copy(Networks_MI[i])
    temp3 = filter_smallest(temp3)
    Networks_MI[i] = temp3



plot_MI(Networks_MI[2])
```

[ ]:
```

```