Applying NetREm model on data where some gene expression nodes are NOT found in the input Protein-Protein Interaction (PPI) network

Understanding: `overlapped_nodes_only` parameter

By: Saniya Khullar

- Please note that in this notebook, we show how NetREm works on data where the gene expression data contains more nodes than those that are found in the input protein-protein interaction (PPI) network.
- Please note that we remove any nodes in the PPI that are not found in the gene expression data because we need gene expression information for all of our predictors (to predict target gene (TG) expression).

Structure of this notebook:

- **Example 1**: We show how NetREm is applied with 6 Transcription Factors (TFs), 5 of which have PPI network input data (missing for TF_6). We will keep all 6 TFs (default: `overlapped_nodes_only = False`) and show how NetREm infers a model (adding default edges between TF_6 and the other 5 predictors).
- **Example 2**: We have the same data (for 6 TFs in gene expression data) as in *Example 1*, but we will instead opt to fit a model only for the 5 TFs (TF_1 to TF_5) found in our input PPI network (`overlapped_nodes_only = True`).
- **Example 3**: We have the gene expression data only for the 5 TFs (TF_1 to TF_5) and we will use `overlapped_nodes_only = False` .
- **Example 4**: We have the gene expression data only for the 5 TFs (TF_1 to TF_5) and we will use `overlapped_nodes_only = True` .

*Please note: we expect to get the **same NetREm model results** (TF-TG regulatory networks given by coefficient c and TF-TF coordination networks of direct/indirect TF-TF interactions given by B) for **Examples 2 to 4**.*

The goal is to build a machine learning model to predict the gene expression levels of our target gene (TG), y, based on the gene expression levels of $N = 6$ candidate Transcription Factors (TF) predictors [TF_1, TF_2, TF_3, TF_4, TF_5, TF_6] in a particular cell-type. Assume the gene expression values for each TF are [X_1, X_2, X_3, X_4, X_5, X_6], respectively. We generate random samples (rows) of data where the Pearson correlations (r) between gene expression of each TF (X) with gene expression of TG as corrVals: [cor(TF_1, y) = 0.9, cor(TF_2, y) = 0.5, cor(TF_3, y) = 0.1, cor(TF_4, y) = -0.2, cor(TF_5, y) = -0.8, cor(TF_6, y) = -0.3].

The dimensions of X are therefore 100,000 rows by 6 columns (predictors).

In [1]:

```
from DemoDataBuilderXandY import generate_dummy_data
from Netrem_model_builder import *
import PriorGraphNetwork as graph
import error_metrics as em
import essential_functions as ef
import netrem_evaluation_functions as nm_eval

# there are 6 entries for corrs_list, corresponding to 6 TF predictors
# correlations of each X predictor TF with y
corrs_list =  [0.9, 0.5, 0.1, -0.2, -0.8, -0.3]

dummy_data_large = generate_dummy_data(corrVals = corrs_list,
                                       num_samples_M = 100000,
                                       train_data_percent = 70)
dummy_data_large
```

```
:) same_train_test_data = False
```

Generating predictors: 6/6 [00:00<00:00, 141.24it/s]

100%

```
Please note that since we hold out 30.0% of our 100000 samples for testin
g, we have:
X_train = 70000 rows (samples) and 6 columns (N = 6 predictors) for traini
ng.
X_test = 30000 rows (samples) and 6 columns (N = 6 predictors) for testin
g.
y_train = 70000 corresponding rows (samples) for training.
y_test = 30000 corresponding rows (samples) for testing.
```

100% 6/6 [00:00<00:00, 329.76it/s]

100% 6/6 [00:00<00:00, 336.25it/s]

100% 6/6 [00:00<00:00, 461.42it/s]

Out[1]:

```
<DemoDataBuilderXandY.DemoDataBuilderXandY at 0x1d10a2b5f60>
```

The X data should be in the form of a Pandas dataframe as below:

In [2]:
```
X_df_large = dummy_data_large.X_df
X_df_large.head()
```

Out[2]:

	TF1	TF2	TF3	TF4	TF5	TF6
0	0.069136	1.154505	0.342957	-1.597101	-0.829135	1.394154
1	-0.020550	0.199960	1.378926	-1.080837	-1.329560	-0.437410
2	-0.572949	-1.228932	-0.579102	-0.155665	2.412702	2.493727
3	0.638589	0.439648	0.758285	1.413038	-0.739683	1.054190
4	-1.244680	-0.937870	0.827576	-1.245802	0.536970	-0.590952

The y data should be in the form of a Pandas dataframe as below:

In [3]:
```
y_df = dummy_data_large.y_df
y_df.head()
```

Out[3]:

	y
0	0.601721
1	1.151619
2	-1.359462
3	0.222055
4	-0.775868

In [4]:
```
# 70,000 samples for training data
# (used to train and fit NetREm model)
X_train_large = dummy_data_large.view_X_train_df()
y_train = dummy_data_large.view_y_train_df()

# 30,000 samples for testing data
X_test_large = dummy_data_large.view_X_test_df()
y_test = dummy_data_large.view_y_test_df()
```

In [5]:
```
1 X_train_large.corr()
```

Out[5]:

	TF1	TF2	TF3	TF4	TF5	TF6
TF1	1.000000	0.452697	0.096390	-0.185516	-0.717721	-0.265090
TF2	0.452697	1.000000	0.052924	-0.102394	-0.401998	-0.148107
TF3	0.096390	0.052924	1.000000	-0.019283	-0.081800	-0.026654
TF4	-0.185516	-0.102394	-0.019283	1.000000	0.159783	0.054849
TF5	-0.717721	-0.401998	-0.081800	0.159783	1.000000	0.233207
TF6	-0.265090	-0.148107	-0.026654	0.054849	0.233207	1.000000

In [6]:
```
1 X_test_large.corr()
```

Out[6]:

	TF1	TF2	TF3	TF4	TF5	TF6
TF1	1.000000	0.449557	0.091530	-0.191634	-0.722327	-0.268918
TF2	0.449557	1.000000	0.045767	-0.102211	-0.404597	-0.154488
TF3	0.091530	0.045767	1.000000	-0.028500	-0.083975	-0.023039
TF4	-0.191634	-0.102211	-0.028500	1.000000	0.166970	0.067488
TF5	-0.722327	-0.404597	-0.083975	0.166970	1.000000	0.232792
TF6	-0.268918	-0.154488	-0.023039	0.067488	0.232792	1.000000

In [7]:
```
training = X_train_large.copy()
training["y"] = y_train
training
```

Out[7]:

	TF1	TF2	TF3	TF4	TF5	TF6	y
0	-0.378744	-0.718500	-1.142852	0.743016	-0.539838	-0.414618	-0.781967
1	-0.724085	-0.331387	-2.260161	-1.388135	0.051536	2.044948	-0.098861
2	-1.979727	-0.145575	0.164909	0.554979	2.047608	-0.175427	-1.358759
3	-0.597911	-0.765671	-1.532276	0.112544	0.448722	1.451416	-1.201094
4	1.943481	1.064808	0.075244	0.043482	-1.734901	-1.169162	1.875530
...	...	...	...	...	...	...	...
69995	-0.958685	-2.424530	0.366898	0.097193	0.202771	-1.029451	-1.315468
69996	0.218740	0.681286	-1.003983	0.826079	-0.049931	-0.178340	0.467944
69997	0.696532	-0.570371	-0.056986	-1.462359	-1.986719	-0.342741	0.502006
69998	-0.122729	1.336861	-0.249281	-0.452302	0.484204	-1.007878	0.277598
69999	0.731626	0.054815	0.073979	-0.610930	-0.780004	-0.303070	0.541264

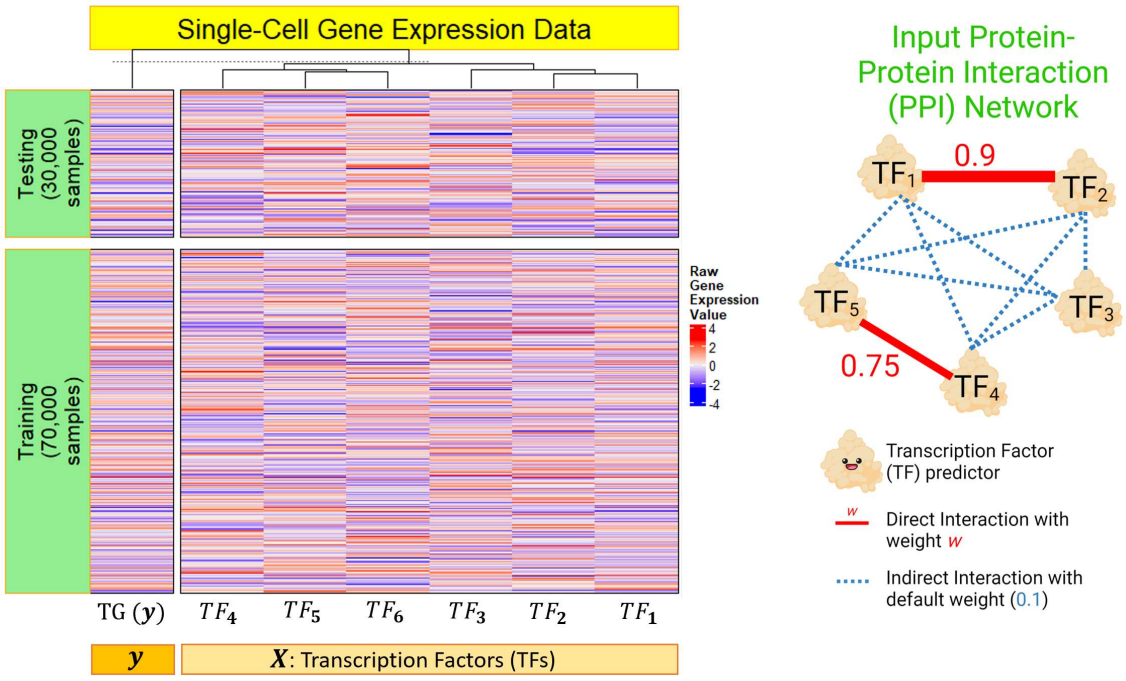
70000 rows × 7 columns

In [8]:
```
testing = X_test_large.copy()
testing["y"] = y_test
testing
```

Out[8]:

	TF1	TF2	TF3	TF4	TF5	TF6	y
0	0.138026	-0.622089	0.158144	0.123349	0.605308	0.066693	-0.368128
1	1.773068	-0.110095	-0.242438	0.508154	-1.047717	-0.589866	2.152347
2	-0.641352	-0.734664	-0.592630	0.218673	0.723363	0.711673	-0.876180
3	1.829922	0.311367	1.729414	-0.052885	-1.052669	-0.389403	1.359079
4	-0.778609	-0.549280	0.796408	-2.338994	-0.597231	-0.991316	-0.450897
...	...	...	...	...	...	...	...
29995	-0.078725	-0.810705	1.036384	-0.544139	0.798243	-0.451409	0.202688
29996	0.798346	-0.445236	-0.064148	0.204764	-1.557834	-0.721080	1.287684
29997	-0.836698	-0.960167	-0.585407	0.205325	1.264368	0.843744	-0.436477
29998	1.385614	0.835561	0.226362	-1.234281	-3.250390	-1.230687	2.022050
29999	0.905795	0.531985	0.164423	-1.363756	-0.620555	-0.511984	0.297392

30000 rows × 7 columns



In [9]:

```
1 X_train_large
```

Out[9]:

	TF1	TF2	TF3	TF4	TF5	TF6
0	-0.378744	-0.718500	-1.142852	0.743016	-0.539838	-0.414618
1	-0.724085	-0.331387	-2.260161	-1.388135	0.051536	2.044948
2	-1.979727	-0.145575	0.164909	0.554979	2.047608	-0.175427
3	-0.597911	-0.765671	-1.532276	0.112544	0.448722	1.451416
4	1.943481	1.064808	0.075244	0.043482	-1.734901	-1.169162
...	...	...	...	...	...	...
69995	-0.958685	-2.424530	0.366898	0.097193	0.202771	-1.029451
69996	0.218740	0.681286	-1.003983	0.826079	-0.049931	-0.178340
69997	0.696532	-0.570371	-0.056986	-1.462359	-1.986719	-0.342741
69998	-0.122729	1.336861	-0.249281	-0.452302	0.484204	-1.007878
69999	0.731626	0.054815	0.073979	-0.610930	-0.780004	-0.303070

70000 rows × 6 columns

In [10]:
```
1 X_test_large
```

Out[10]:

	TF1	TF2	TF3	TF4	TF5	TF6
0	0.138026	-0.622089	0.158144	0.123349	0.605308	0.066693
1	1.773068	-0.110095	-0.242438	0.508154	-1.047717	-0.589866
2	-0.641352	-0.734664	-0.592630	0.218673	0.723363	0.711673
3	1.829922	0.311367	1.729414	-0.052885	-1.052669	-0.389403
4	-0.778609	-0.549280	0.796408	-2.338994	-0.597231	-0.991316
...	...	...	...	...	...	...
29995	-0.078725	-0.810705	1.036384	-0.544139	0.798243	-0.451409
29996	0.798346	-0.445236	-0.064148	0.204764	-1.557834	-0.721080
29997	-0.836698	-0.960167	-0.585407	0.205325	1.264368	0.843744
29998	1.385614	0.835561	0.226362	-1.234281	-3.250390	-1.230687
29999	0.905795	0.531985	0.164423	-1.363756	-0.620555	-0.511984

30000 rows × 6 columns

In [11]:
```
1 edge_list = [["TF1", "TF2", 0.9], ["TF4", "TF5", 0.75],
2              ["TF1", "TF3"], ["TF1", "TF4"], ["TF1", "TF5"],
3              ["TF2", "TF3"], ["TF2", "TF4"], ["TF2", "TF5"],
4              ["TF3", "TF4"], ["TF3", "TF5"]]
5 beta_network_val = 1
```

In [12]:

```
# for reference:
# this is what the edge_list looks like, for only TFs 1 to 5.
pd.DataFrame(edge_list)
# data is missing for TF6 for this input Protein-Protein Interaction (P
```

Out[12]:

	0	1	2
0	TF1	TF2	0.90
1	TF4	TF5	0.75
2	TF1	TF3	NaN
3	TF1	TF4	NaN
4	TF1	TF5	NaN
5	TF2	TF3	NaN
6	TF2	TF4	NaN
7	TF2	TF5	NaN
8	TF3	TF4	NaN
9	TF3	TF5	NaN

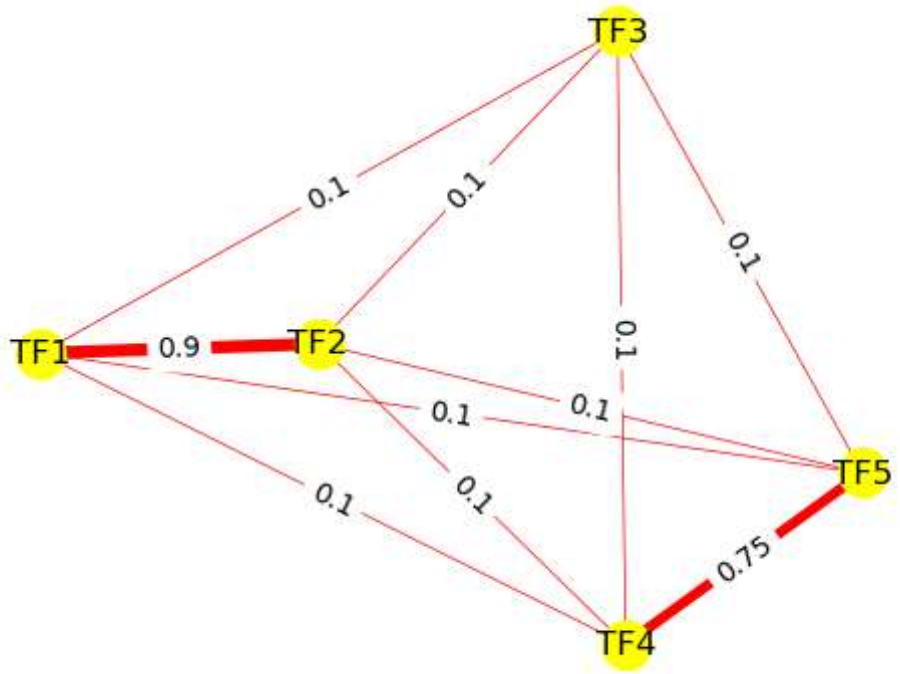
Example 1:

Include all 6 TFs found in the gene expression data (even though TF6 is not found in the input PPI). The default is that `overlapped_nodes_only = False`

In [13]:

```
# Building the network regularized regression model:
# Please note: To include nodes found in the gene expression data
# that are not found in the PPI Network (e.g. TF6 in our case),
# we use False for the overlapped_nodes_only argument (otherwise,
# we would only use TFs 1 to 5).
# By default, edges are constructed between all of the nodes;
# nodes with a missing edge are assigned the default_edge_weight.
# by default: overlapped_nodes_only = False, so we include ALL 6 TFs
# found in the gene expression data
netrem_demo_large = netrem(edge_list = edge_list,
                           beta_net = beta_network_val,
                           model_type = "LassoCV",
                           view_network = True)

# Fitting the NetREm model on training data: X_train and y_train:
netrem_demo_large.fit(X_train_large, y_train)
```

```
using beta_net default of 1
Please note that we need to update the network information
```

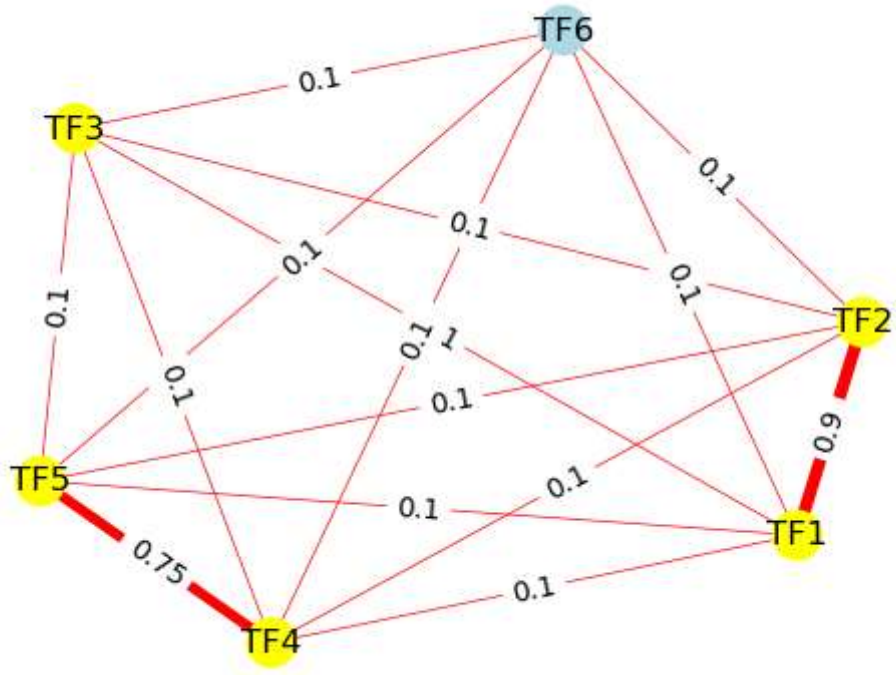


:) 1 new nodes added to network based on gene expression data ['TF6']

Out[13]:

NetREmModel

```
NetREmModel(verbose=False, overlapped_nodes_only=False, num_cv_folds=5, n
um_jobs=-1, all_pos_coefs=False, model_type=LassoCV, standardize_X=True,
center_y=True, use_network=True, y_intercept=False, max_lasso_iterations=
10000, view_network=True, tolerance=0.0001, lasso_selection=cyclic, lasso
cv_eps=0.001, lassocv_n_alphas=100, lassocv_alphas=None, network=<PriorGr
aphNetwork.PriorGraphNetwork object at 0x000001D112E75540>, alpha_lasso=L
assoCV finds optimal alpha)
```



In [14]:

```
1 # TF-TG regulatory network coefficients c
2 netrem_demo_large.model_coef_df
```

Out[14]:

	y_intercept	TF1	TF2	TF3	TF4	TF5	TF6
0	None	0.580109	0.132124	0.002992	-0.038524	-0.298227	0.001154

In [15]:
```
# TF-TF coordination network coordination scores B
netrem_demo_large.B_interaction_df
```

Out[15]:

	TF1	TF2	TF3	TF4	TF5	TF6
TF1	1.072150	0.402747	-0.079204	-0.191066	-0.723271	-0.440684
TF2	0.402747	1.072150	-0.122671	-0.107944	-0.407548	-0.323701
TF3	-0.079204	-0.122671	28.777778	-0.194877	-0.257394	-5.582210
TF4	-0.191066	-0.107944	-0.194877	1.063825	0.118158	-0.120745
TF5	-0.723271	-0.407548	-0.257394	0.118158	1.063825	0.057612
TF6	-0.440684	-0.323701	-5.582210	-0.120745	0.057612	28.777778

In [16]:
```
pred_y_test = netrem_demo_large.predict(X_test_large) # predicted value
mse_test = netrem_demo_large.test_mse(X_test_large, y_test)
print(f"The testing Mean Square Error (MSE) is {mse_test}")
```

```
The testing Mean Square Error (MSE) is 0.13716012957807497
```

In [17]:
```
netrem_demo_large.final_corr_vs_coef_df
```

Out[17]:

	info	input_data	TF1	TF2	TF3	TF4	TF5	TF6
0	network regression coeff. with y: y	X_train	0.580109	0.132124	0.002992	-0.038524	-0.298227	0.001154
0	corr (r) with y: y	X_train	0.900365	0.50225	0.102892	-0.20339	-0.798606	-0.294202
0	Absolute Value NetREm Coefficient Ranking	X_train	1	3	5	4	2	6

Example 2:

Focus only on the 5 common TFs between Network and the Gene Expression Data: `overlapped_nodes_only = True`

In [18]:
```
1 X_train_large.head()
```

Out[18]:

	TF1	TF2	TF3	TF4	TF5	TF6
0	-0.378744	-0.718500	-1.142852	0.743016	-0.539838	-0.414618
1	-0.724085	-0.331387	-2.260161	-1.388135	0.051536	2.044948
2	-1.979727	-0.145575	0.164909	0.554979	2.047608	-0.175427
3	-0.597911	-0.765671	-1.532276	0.112544	0.448722	1.451416
4	1.943481	1.064808	0.075244	0.043482	-1.734901	-1.169162

In [19]:

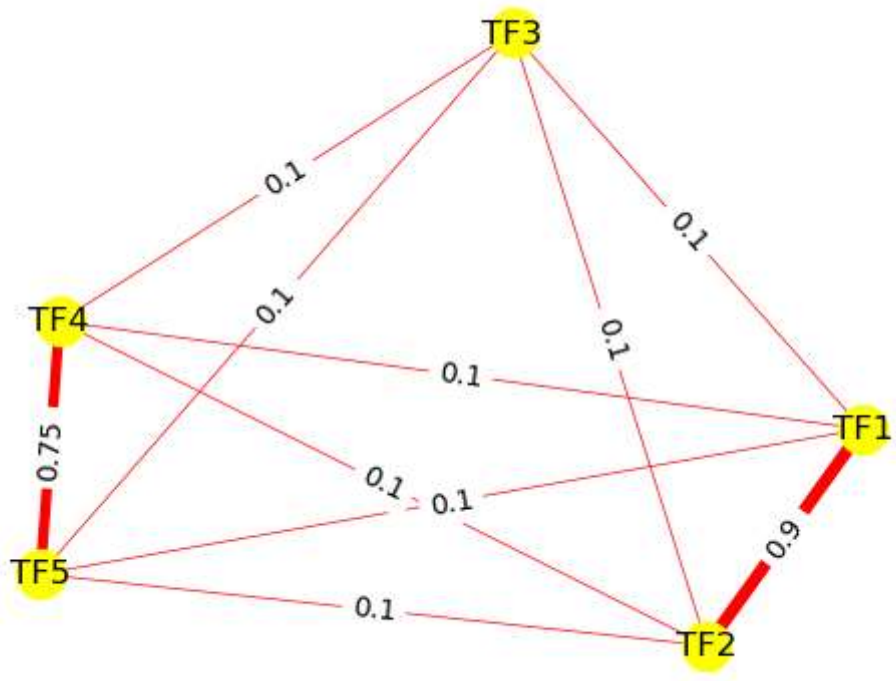
```
# Building the network regularized regression model:
# Please note: To include nodes found in the gene expression data
# that are not found in the PPI Network
# (e.g. TF6 in our case), we use False for the overlapped_nodes_only
# argument (otherwise, we would only use TFs 1 to 5).
# By default, edges are constructed between all of the nodes;
# nodes with a missing edge are assigned the default_edge_weight.
netrem_demo_small_v1 = netrem(edge_list = edge_list,
                              beta_net = beta_network_val,
                              model_type = "LassoCV",
                              overlapped_nodes_only = True, # we only use TFs 1 to 5
                                                            # since those are found in
                                                            # gene expression & PPI
                              view_network = True)

# Fitting the NetREm model on training data: X_train and y_train:
netrem_demo_small_v1.fit(X_train_large, y_train)
```

```
using beta_net default of 1
```

Out[19]:

NetREmModel

```
NetREmModel(verbose=False, overlapped_nodes_only=True, num_cv_folds=5, nu
m_jobs=-1, all_pos_coefs=False, model_type=LassoCV, standardize_X=True, c
enter_y=True, use_network=True, y_intercept=False, max_lasso_iterations=1
0000, view_network=True, tolerance=0.0001, lasso_selection=cyclic, lassoc
v_eps=0.001, lassocv_n_alphas=100, lassocv_alphas=None, network=<PriorGra
phNetwork.PriorGraphNetwork object at 0x000001D1161C9B70>, alpha_lasso=La
ssoCV finds optimal alpha)
```



In [20]:
```
1 netrem_demo_small_v1.model_coef_df
```

Out[20]:

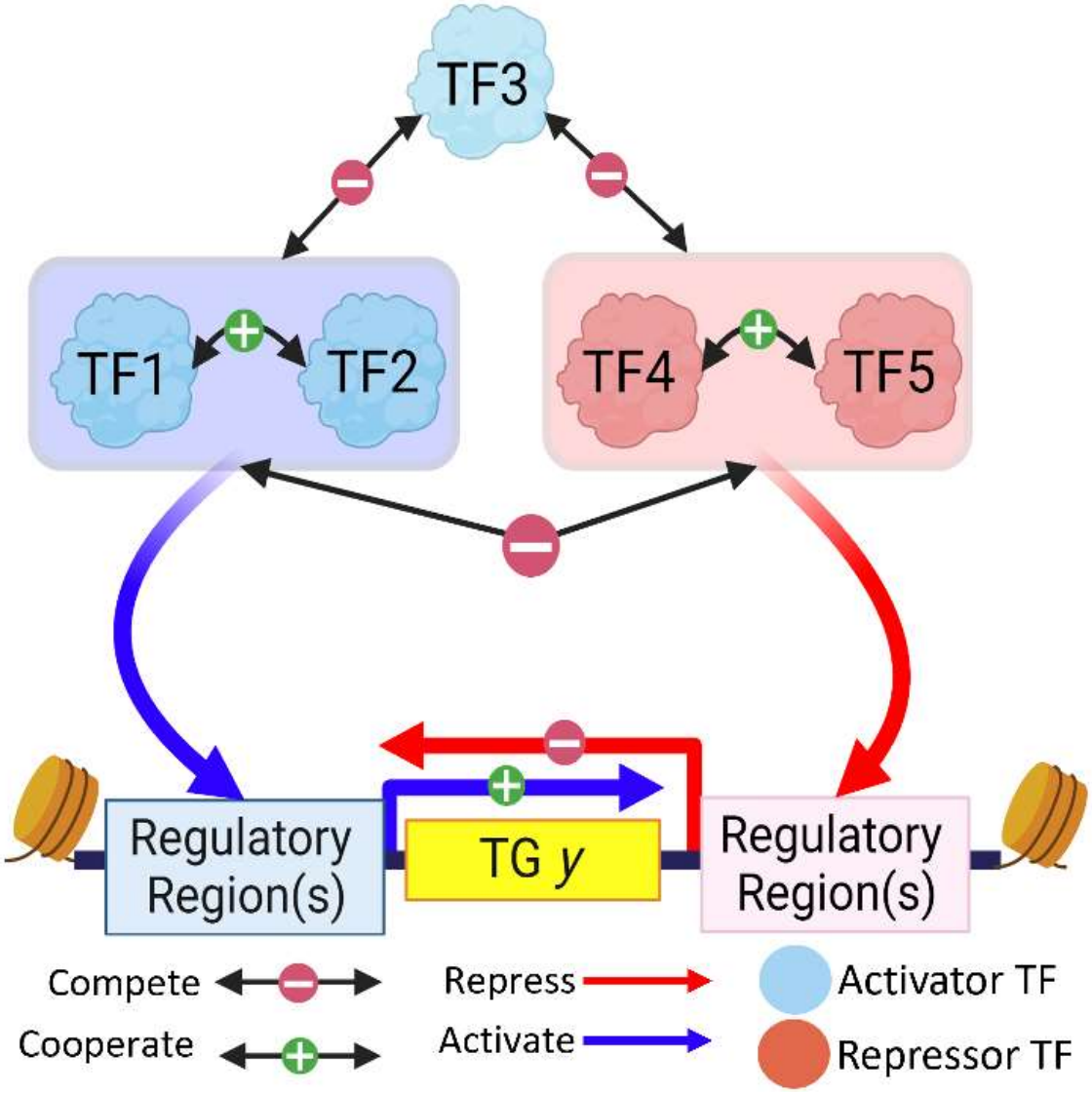
	y_intercept	TF1	TF2	TF3	TF4	TF5
0	None	0.565498	0.145701	0.003299	-0.042545	-0.295653

In [21]:
```
1 netrem_demo_small_v1.B_interaction_df
```

Out[21]:

	TF1	TF2	TF3	TF4	TF5
TF1	1.095904	0.380769	-0.156466	-0.193508	-0.725713
TF2	0.380769	1.095904	-0.199932	-0.110386	-0.409990
TF3	-0.156466	-0.199932	33.000000	-0.272139	-0.334656
TF4	-0.193508	-0.110386	-0.272139	1.083916	0.099843
TF5	-0.725713	-0.409990	-0.334656	0.099843	1.083916

In the context of gene regulation, our results may thereby be interpreted in the following way:



In [22]:

```
# predicted values for y_test:
pred_y_test = netrem_demo_small_v1.predict(X_test_large)
mse_test = netrem_demo_small_v1.test_mse(X_test_large, y_test)
print(f"The testing Mean Square Error (MSE) is {mse_test}")
netrem_demo_small_v1.final_corr_vs_coef_df
```

```
The testing Mean Square Error (MSE) is 0.13896503741769764
```

Out[22]:

	info	input_data	TF1	TF2	TF3	TF4	TF5
0	network regression coeff. with y: y	X_train	0.565498	0.145701	0.003299	-0.042545	-0.295653
0	corr (r) with y: y	X_train	0.900365	0.50225	0.102892	-0.20339	-0.798606
0	Absolute Value NetREm Coefficient Ranking	X_train	1	3	5	4	2

Example 3:

We illustrate that the results for `overlapped_nodes_only = False` are the same if we use the smaller dataset with 5 TFs

In [23]:
```
X_df_small = dummy_data_large.X_df.drop(columns
                                        = ["TF6"])
X_df_small.head()
```

Out[23]:

	TF1	TF2	TF3	TF4	TF5
0	0.069136	1.154505	0.342957	-1.597101	-0.829135
1	-0.020550	0.199960	1.378926	-1.080837	-1.329560
2	-0.572949	-1.228932	-0.579102	-0.155665	2.412702
3	0.638589	0.439648	0.758285	1.413038	-0.739683
4	-1.244680	-0.937870	0.827576	-1.245802	0.536970

In [24]:
```
# We remove TF6 from the gene expression data to
# pretend we did not have TF6 information
X_train_small = dummy_data_large.view_X_train_df().drop(columns
                                                        = ["TF6"])

X_test_small = dummy_data_large.view_X_test_df().drop(columns
                                                      = ["TF6"])
```

In [25]:
```
1 X_train_small
```

Out[25]:

	TF1	TF2	TF3	TF4	TF5
0	-0.378744	-0.718500	-1.142852	0.743016	-0.539838
1	-0.724085	-0.331387	-2.260161	-1.388135	0.051536
2	-1.979727	-0.145575	0.164909	0.554979	2.047608
3	-0.597911	-0.765671	-1.532276	0.112544	0.448722
4	1.943481	1.064808	0.075244	0.043482	-1.734901
...	...	...	...	...	...
69995	-0.958685	-2.424530	0.366898	0.097193	0.202771
69996	0.218740	0.681286	-1.003983	0.826079	-0.049931
69997	0.696532	-0.570371	-0.056986	-1.462359	-1.986719
69998	-0.122729	1.336861	-0.249281	-0.452302	0.484204
69999	0.731626	0.054815	0.073979	-0.610930	-0.780004

70000 rows × 5 columns

In [26]:
```
1 X_test_small
```

Out[26]:

	TF1	TF2	TF3	TF4	TF5
0	0.138026	-0.622089	0.158144	0.123349	0.605308
1	1.773068	-0.110095	-0.242438	0.508154	-1.047717
2	-0.641352	-0.734664	-0.592630	0.218673	0.723363
3	1.829922	0.311367	1.729414	-0.052885	-1.052669
4	-0.778609	-0.549280	0.796408	-2.338994	-0.597231
...	...	...	...	...	...
29995	-0.078725	-0.810705	1.036384	-0.544139	0.798243
29996	0.798346	-0.445236	-0.064148	0.204764	-1.557834
29997	-0.836698	-0.960167	-0.585407	0.205325	1.264368
29998	1.385614	0.835561	0.226362	-1.234281	-3.250390
29999	0.905795	0.531985	0.164423	-1.363756	-0.620555

30000 rows × 5 columns

In [27]:

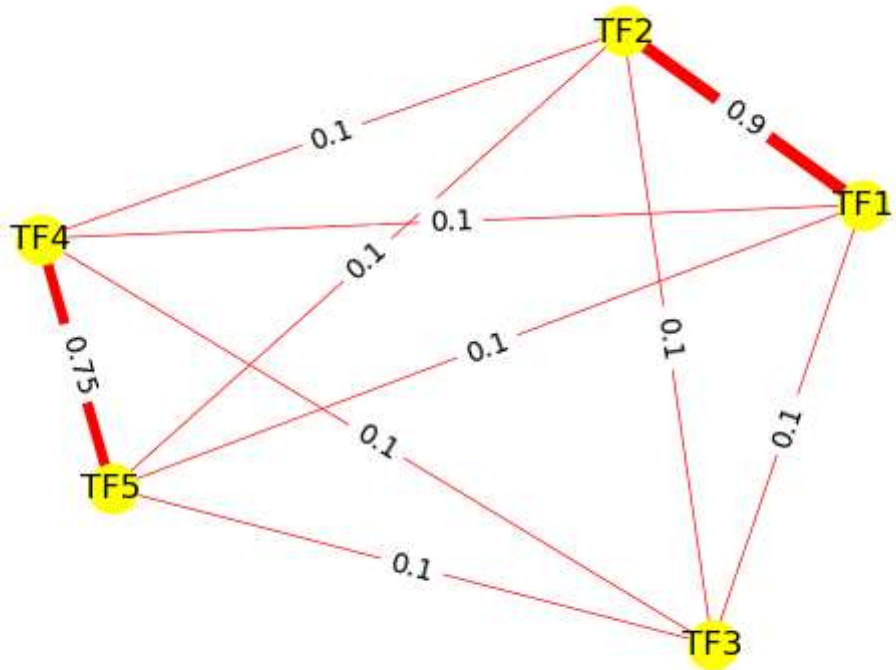
```
# Building the network regularized regression model:
# Please note: To include nodes found in the gene expression data
# that are not found in the PPI Network, we use False for
# the overlapped_nodes_only argument.
# default: overlapped_nodes_only = False
# By default, edges are constructed between all of the nodes;
# nodes with a missing edge are assigned the default_edge_weight.
netrem_demo_small_v2 = netrem(edge_list = edge_list,
                beta_net = beta_network_val,
              model_type = "LassoCV",
                view_network = True)

# Fitting the NetREm model on training data: X_train and y_train:
netrem_demo_small_v2.fit(X_train_small, y_train)
```

```
using beta_net default of 1
```

Out[27]:

NetREmModel

```
NetREmModel(verbose=False, overlapped_nodes_only=False, num_cv_folds=5, n
um_jobs=-1, all_pos_coefs=False, model_type=LassoCV, standardize_X=True,
center_y=True, use_network=True, y_intercept=False, max_lasso_iterations=
10000, view_network=True, tolerance=0.0001, lasso_selection=cyclic, lasso
cv_eps=0.001, lassocv_n_alphas=100, lassocv_alphas=None, network=<PriorGr
aphNetwork.PriorGraphNetwork object at 0x000001D1166D5E70>, alpha_lasso=L
assoCV finds optimal alpha)
```



In [28]:
```
1 netrem_demo_small_v2.B_interaction_df
```

Out[28]:

	TF1	TF2	TF3	TF4	TF5
TF1	1.095904	0.380769	-0.156466	-0.193508	-0.725713
TF2	0.380769	1.095904	-0.199932	-0.110386	-0.409990
TF3	-0.156466	-0.199932	33.000000	-0.272139	-0.334656
TF4	-0.193508	-0.110386	-0.272139	1.083916	0.099843
TF5	-0.725713	-0.409990	-0.334656	0.099843	1.083916

In [29]:
```
1 # predicted values for y_test
2 pred_y_test = netrem_demo_small_v2.predict(X_test_small)
3 mse_test = netrem_demo_small_v2.test_mse(X_test_small, y_test)
4 print(f"The testing Mean Square Error (MSE) is {mse_test}")
5 netrem_demo_small_v2.final_corr_vs_coef_df
```

```
The testing Mean Square Error (MSE) is 0.13896503741769764
```

Out[29]:

	info	input_data	TF1	TF2	TF3	TF4	TF5
0	network regression coeff. with y: y	X_train	0.565498	0.145701	0.003299	-0.042545	-0.295653
0	corr (r) with y: y	X_train	0.900365	0.50225	0.102892	-0.20339	-0.798606
0	Absolute Value NetREm Coefficient Ranking	X_train	1	3	5	4	2

In [30]:
```
1 netrem_demo_small_v2.A_df
```

Out[30]:

	TF1	TF2	TF3	TF4	TF5
TF1	1.198801	-0.899101	-3.160698	-0.099900	-0.099900
TF2	-0.899101	1.198801	-3.160698	-0.099900	-0.099900
TF3	-3.160698	-3.160698	400.000000	-3.160698	-3.160698
TF4	-0.099900	-0.099900	-3.160698	1.048951	-0.749251
TF5	-0.099900	-0.099900	-3.160698	-0.749251	1.048951

In [31]:
```
1 X_test_small
```

Out[31]:

	TF1	TF2	TF3	TF4	TF5
0	0.138026	-0.622089	0.158144	0.123349	0.605308
1	1.773068	-0.110095	-0.242438	0.508154	-1.047717
2	-0.641352	-0.734664	-0.592630	0.218673	0.723363
3	1.829922	0.311367	1.729414	-0.052885	-1.052669
4	-0.778609	-0.549280	0.796408	-2.338994	-0.597231
...	...	...	...	...	...
29995	-0.078725	-0.810705	1.036384	-0.544139	0.798243
29996	0.798346	-0.445236	-0.064148	0.204764	-1.557834
29997	-0.836698	-0.960167	-0.585407	0.205325	1.264368
29998	1.385614	0.835561	0.226362	-1.234281	-3.250390
29999	0.905795	0.531985	0.164423	-1.363756	-0.620555

30000 rows × 5 columns

Example 4:

We illustrate that the results for `overlapped_nodes_only = True` are the same as for `overlapped_nodes_only = False` for datasets where all nodes in the gene expression data are found in the input PPI network.

Here, all 5 gene expression nodes (TF_1, TF_2, TF_3, TF_4, TF_5) are found in the PPI network so the Boolean (True/False) value for `overlapped_nodes_only` will NOT matter :)

In [32]:

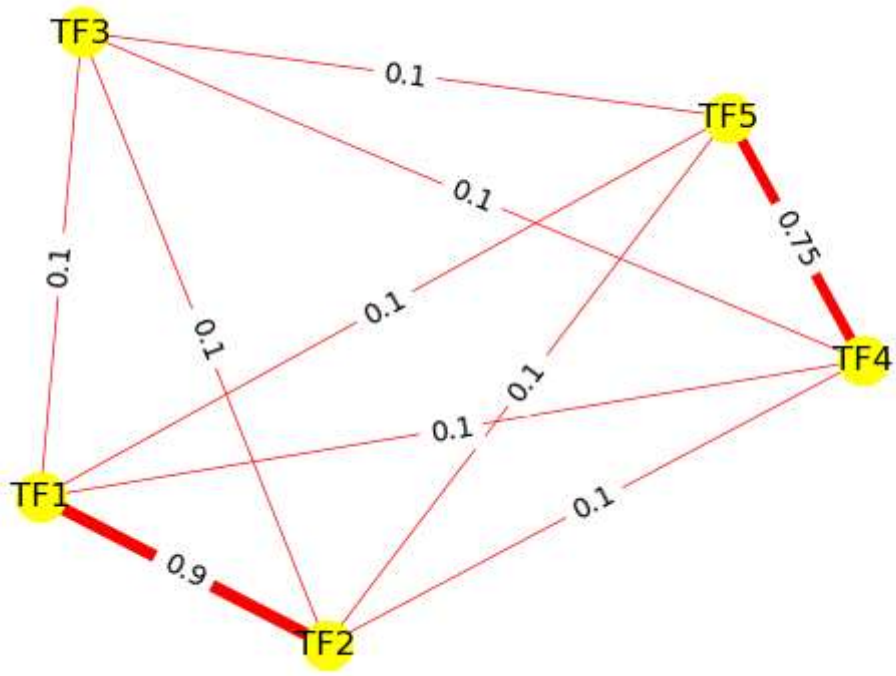
```
# Building the network regularized regression model:
# Please note: To include nodes found in the gene expression data
# that are not found in the PPI Network:
# we use False for the overlapped_nodes_only argument.
# Here, all 5 gene expression nodes are found in the PPI network,
# so overlapped_nodes_only makes no difference.
# By default, edges are constructed between all of the nodes;
# nodes with a missing edge are assigned the default_edge_weight.
netrem_demo_small_v3 = netrem(edge_list = edge_list,
                              beta_net = beta_network_val,
                             overlapped_nodes_only = True,
                            model_type = "LassoCV",
                              view_network = True)

# Fitting the NetREm model on training data: X_train and y_train:
netrem_demo_small_v3.fit(X_train_small, y_train)
```

```
using beta_net default of 1
```

Out[32]:

NetREmModel

```
NetREmModel(verbose=False, overlapped_nodes_only=True, num_cv_folds=5, nu
m_jobs=-1, all_pos_coefs=False, model_type=LassoCV, standardize_X=True, c
enter_y=True, use_network=True, y_intercept=False, max_lasso_iterations=1
0000, view_network=True, tolerance=0.0001, lasso_selection=cyclic, lassoc
v_eps=0.001, lassocv_n_alphas=100, lassocv_alphas=None, network=<PriorGra
phNetwork.PriorGraphNetwork object at 0x000001D1166D7640>, alpha_lasso=La
ssoCV finds optimal alpha)
```



In [33]:
```
1 netrem_demo_small_v3.B_interaction_df
```

Out[33]:

	TF1	TF2	TF3	TF4	TF5
TF1	1.095904	0.380769	-0.156466	-0.193508	-0.725713
TF2	0.380769	1.095904	-0.199932	-0.110386	-0.409990
TF3	-0.156466	-0.199932	33.000000	-0.272139	-0.334656
TF4	-0.193508	-0.110386	-0.272139	1.083916	0.099843
TF5	-0.725713	-0.409990	-0.334656	0.099843	1.083916

In [34]:
```
1 # predicted values for y_test
2 pred_y_test = netrem_demo_small_v3.predict(X_test_small)
3 mse_test = netrem_demo_small_v3.test_mse(X_test_small, y_test)
4 print(f"The testing Mean Square Error (MSE) is {mse_test}")
5 netrem_demo_small_v3.final_corr_vs_coef_df
```

```
The testing Mean Square Error (MSE) is 0.13896503741769764
```

Out[34]:

	info	input_data	TF1	TF2	TF3	TF4	TF5
0	network regression coeff. with y: y	X_train	0.565498	0.145701	0.003299	-0.042545	-0.295653
0	corr (r) with y: y	X_train	0.900365	0.50225	0.102892	-0.20339	-0.798606
0	Absolute Value NetREm Coefficient Ranking	X_train	1	3	5	4	2

In [35]:
```
1 netrem_demo_small_v3.A_df
```

Out[35]:

	TF1	TF2	TF3	TF4	TF5
TF1	1.198801	-0.899101	-3.160698	-0.099900	-0.099900
TF2	-0.899101	1.198801	-3.160698	-0.099900	-0.099900
TF3	-3.160698	-3.160698	400.000000	-3.160698	-3.160698
TF4	-0.099900	-0.099900	-3.160698	1.048951	-0.749251
TF5	-0.099900	-0.099900	-3.160698	-0.749251	1.048951