

NBS-Predict Tutorial

Brief Outline

- **Aim:** To evaluate the performance of NBS-Predict and the alternative algorithms in predicting target variables and identifying edges with ground truth on simulated network data.
- **Method:**
 - The alternative algorithms were the Elastic Net, Lasso, p-value thresholding, Top 5% of features, and Connectome-based predictive modeling (Shen et al., 2017).
 - Two sets of scale-free synthetic networks were generated (for regression and classification tasks). The scale free networks were generated using Barabási-Albert model (Albert & Barabási, 2002).
 - In the classification task, 100 observations were generated (50 observations each for control and contrast groups), and CNR values of 0.25, 0.50, 0.75, and 1.0 were added to 50 edges that form connected component in the contrast group (shown in Zalesky et al., 2010).
 - In the regression task, 250 observations were generated, and the noise values of 0.1, 1, 3, and 5 were added to the target variables that were generated using coefficients of 50 edges that form connected component.
 - Each algorithm performed in a 10-repeated 10-fold CV structure.
 - The total number of simulation iteration was 1000.
 - Accuracy and correlation coefficient metrics were used to evaluate the target prediction performance of the algorithms in the classification and regression tasks, respectively. The performance of algorithms in identifying edges with ground truth was evaluated using ROC curves.

Simulation

Type “`sim_testNBSPredict(parameters)`”

to run the simulation.

It requires several parameters. To check the available parameters, type “`help sim_testNBSPredict`”. It will return the long list of available parameters and their brief descriptions.

```
Command Window
>> help sim_testNBSPredict
sim_testNBSPredict performs simulation using simulated network data.
It is used to test prediction performance of the NBS-Predict and
alternative algorithms (elastic net, lasso, p-value thresholding, top 5%,
and connectome based predictive modeling (Shen et al., 2017)) on the
the simulated network data (small world, scale free or random).

Input:
simIter = Total number of iteration (default = 1000). Please set a
number that is multiplier of nCores to make sure that the desired
number of cores will be utilized in all iterations. Total number of
iteration will be the multiplier of the nCores even if you set
another numbers. For example:
    if simIter = 1001; and nCores = 10;
    It will run 1000 times instead of 1001.
nCores = Number of cores to be used, enter -1 to utilize all cores
(default = 1).
conds = Structure of effect/noise conditions. The structure must be
as followings: conds.(condName) = condValue. Defaults:
    Classification - 0.25, 0.5, 0.75, 1.0 CNRs
    Regression     - 0.1, 1.0, 3.0, 5.0 Noise
ifRegression = Runs simulations for classification or regression
problems (1 Regression, 0 Classification) (default = 0).
randSeed = Seed number for random generator. Pass an integer for
reproducible results or 'shuffle' to randomize (default = 42).
algorithm = Algorithms used to simulate (default = NBSPredict):
    NBSPredict = NBS-Predict.
    CPM        = Connectome-based predictive modeling (Shen et al., 2017)
    ElasticNet = Elastic Net.
    Lasso      = Lasso.
    pVal       = Selects suprathreshold features below a given p-value
                threshold.
    Top5       = Selects top 5% of features based on their test
                statistics.
saveX = Whether features used in each iteration will be saved
(default = 1).
formNetwork = Whether edges with ground truth form a network
(default = 1).

Additional arguments (e.g. networks, models parameters) documented in
test_NBSPredict function.

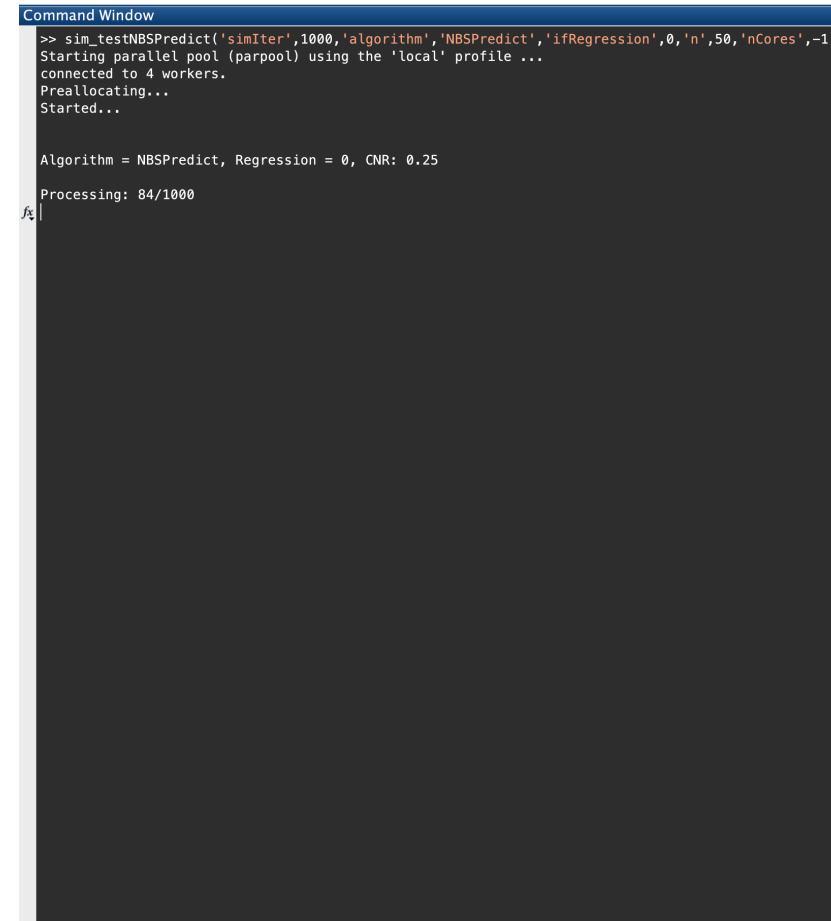
Output:
simResults = Output structure as follows:
    SimResults:
        info = Several information and parameters.
        conds: Effect/Noise conditions
            cond (e.g. CNR_25 for 0.25 CNR):
                y_true = True labels.
                edgeWeight = Vectorized raw edge weights.
                sEdgeWeight = Vectorized scaled edge weights.
                predPerf = Prediction performances.
```

Simulation

Let's simulate the performance of NBS-Predict in classification tasks using simulated networks.

To do that, type:

```
sim_testNBSPredict('simIter', 1000, 'algorithm', 'NBSPredict',  
'ifRegression', 0, 'n', 50, 'nCores', -1)
```



```
Command Window
>> sim_testNBSPredict('simIter',1000,'algorithm','NBSPredict','ifRegression',0,'n',50,'nCores',-1)
Starting parallel pool (parpool) using the 'local' profile ...
connected to 4 workers.
Preallocating...
Started...

Algorithm = NBSPredict, Regression = 0, CNR: 0.25
Processing: 84/1000
```

Simulation

This function, in each iteration, simply generates 100x100 scale-free networks comprising of 1000 edges. Of those edges, 50 are contrasted using contrast to noise ratio (CNR) values of 0.25, 0.50, 0.75, and 1.0.

NBS-Predict (10-repeated 10-fold CV) runs 1000 times.

If you want to use another CNR values, you can simply create a structure comprising CNR conditions.

```
Command Window
>> conds.zeroOne = 0.1;
>> conds.zeroFive = 0.5;
>> conds.zeroEight = 0.8;
>> conds

conds =

struct with fields:

    zeroOne: 0.1000
    zeroFive: 0.5000
    zeroEight: 0.8000

fx >> sim_testNBSPredict('simIter',1000,'algorithm','NBSPredict','conds',conds,'ifRegression',0,'n',50,'nCores',-1)
```

Simulation

After running the command, you can check the progress of the simulation.

```
Command Window
>> sim_testNBSPredict('simIter',1000,'algorithm','NBSPredict','ifRegression',0,'n',50,'nCores',-1)
Starting parallel pool (parpool) using the 'local' profile ...
connected to 4 workers.
Preallocating...
Started...

Algorithm = NBSPredict, Regression = 0, CNR: 0.25
Processing: 84/1000
```

Simulation

Let's run similar simulations for the regression task.

To do that, type:

```
sim_testNBSPredict('simIter', 1000, 'algorithm', 'NBSPredict',  
'ifRegression', 1, 'n', 125, 'nCores', -1)
```

This time we increased the number of observation pairs from 50 to 125 since regression problems require more observation.

By default, this function, in each iteration, generates similar scale-free networks and adds noise values of 0.1, 1, 3, and 5 to the target variable generated using 50 edges that form the connected component.

```
Command Window
>> sim_testNBSPredict('simIter',1000,'algorithm','NBSPredict','ifRegression',1,'n',125,'nCores',-1)
Starting parallel pool (parpool) using the 'local' profile ...
connected to 4 workers.
Preallocating...
Started...

Algorithm = NBSPredict, Regression = 1, Noise: 0.10
Processing: 64/1000
```

Simulation

To test alternative algorithms, simply change the algorithm parameter with the name of the desired algorithm.

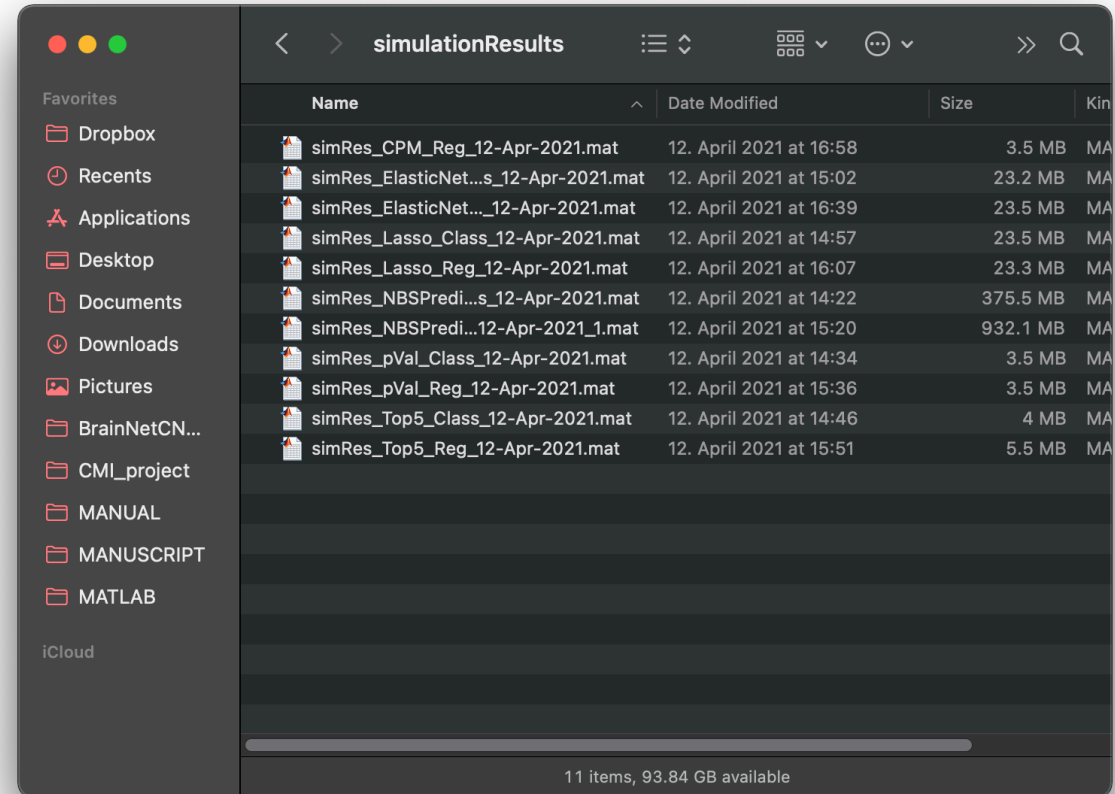
Available algorithms are documented in the function.

```
Command Window
>> sim_testNBSPredict('simIter',1000,'algorithm','ElasticNet','ifRegression',1,'n',50,'nCores',-1)
Parallel pool using the 'local' profile is shutting down.
Starting parallel pool (parpool) using the 'local' profile ...
connected to 4 workers.
Preallocating...
Started...

Algorithm = ElasticNet, Regression = 1, Noise: 0.10
Processing: 36/1000
```


Simulation

After the simulation is done, the results will be saved into the “~/NBSPredict/simulationResults/” directory.



Prepare Data for Plotting

We can plot the results using the functions from NBS-Predict!

To do that, we first need to prepare the results structures using the “`prep_plotData`” function.

This function computes the algorithms’ performance (TPR, FPR, and more) in identifying edges with ground truth. For each simulation iteration, it simply selects sets of edges using several edge weight thresholds and computes performance metrics.

```
>> help prep_plotData
prep_plotData prepares data for simulation plots. Data can be used by
gen_SimFigures function to plot results. The files generated using this
function can be used by prep_plotData function.

Input:
  Simulation results derived using sim_testNBSPredict function
    (e.g. simRes_CPM_Reg_09-Apr-2021.mat).
  metrics = Cell array including performance metrics
    (default = {'sensitivity','specificity'}). Available metrics are
    documented in compute_modelMetrics function.
  wtSteps = Weight threshold steps between 0 and 1 (default = 100).
  nCores  = Number of CPU cores to be used (default = 1).
  ifSave  = Save processed data to generate plots (default = 1). If true
    it will save processed file into
    ~/plotData/[inputFileName]_plotData.mat

Output:
  plotData = Structure including results of simulation for NBSPredict.

Example:
  [plotData] = prep_plotData;
  [plotData] = prep_plotData('wtSteps',50,'nCores',4,...
    'metrics',{'precision','recall'});

Created by Emin Serin, 11.04.2021

See also: sim\_testNBSPredict, plot\_plotData, compute\_modelMetrics
```

Prepare Data for Plotting

By default, it calculates TPR and FPR to generate ROC curves. But additional metrics (e.g. precision, recall, AUC) can be used by simply using “metrics” parameters.

The available metrics are documented in the “compute_modelMetrics” function!

```
>> help prep_plotData
prep_plotData prepares data for simulation plots. Data can be used by
gen_SimFigures function to plot results. The files generated using this
function can be used by prep_plotData function.

Input:
  Simulation results derived using sim_testNBSPredict function
  (e.g. simRes_CPM_Reg_09-Apr-2021.mat).
  metrics = Cell array including performance metrics
  (default = {'sensitivity','specificity'}). Available metrics are
  documented in compute_modelMetrics function.
  wtSteps = Weight threshold steps between 0 and 1 (default = 100).
  nCores = Number of CPU cores to be used (default = 1).
  ifSave = Save processed data to generate plots (default = 1). If true
  it will save processed file into
  ~/plotData/[inputFileName]_plotData.mat

Output:
  plotData = Structure including results of simulation for NBSPredict.

Example:
  [plotData] = prep_plotData;
  [plotData] = prep_plotData('wtSteps',50,'nCores',4,...
  'metrics',{'precision','recall'});

Created by Emin Serin, 11.04.2021

See also: sim\_testNBSPredict, plot\_plotData, compute\_modelMetrics
```

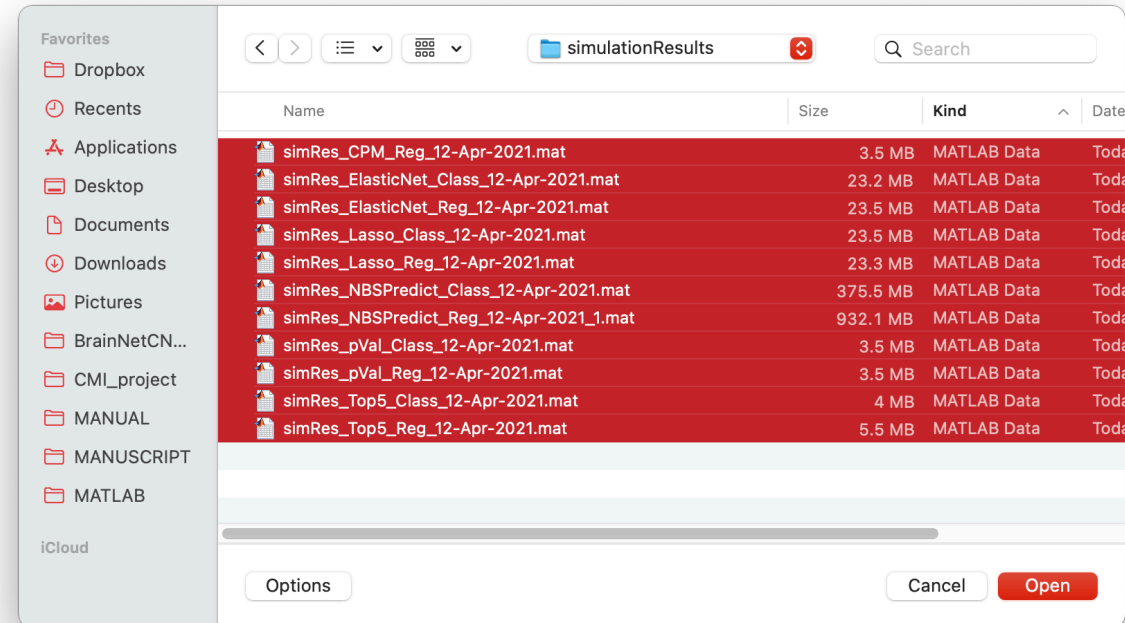
Prepare Data for Plotting

To run the function, simply type:

`“prep_plotData”`

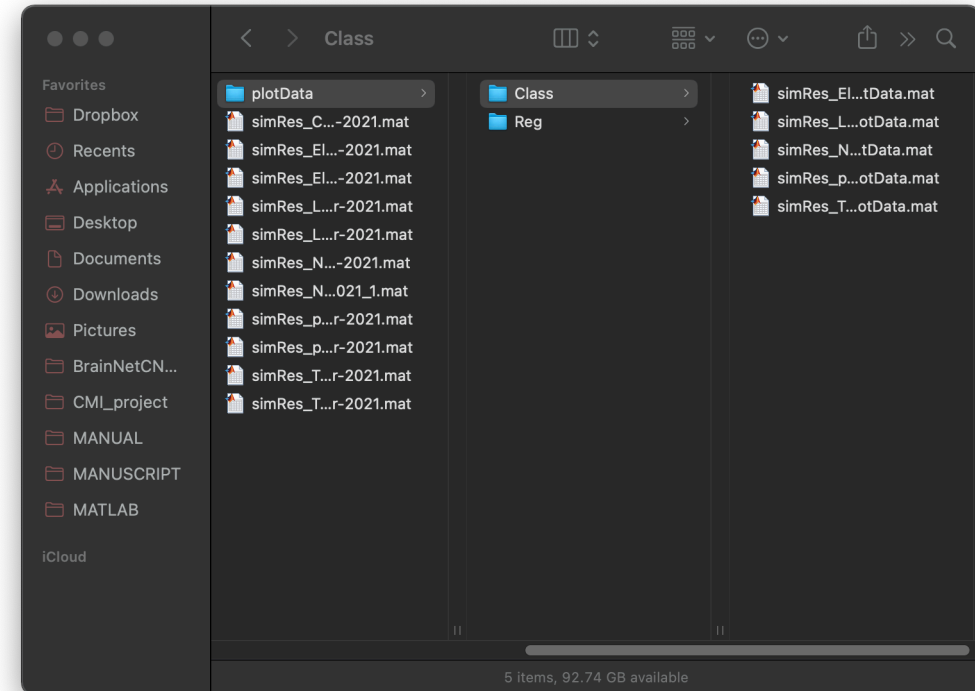
A window will pop up and ask to load results files generated after the simulation.

Select all the files simply to process them recursively.



Prepare Data for Plotting

After processing, the processed files will be saved in the “plotData” folder. Results for classification and regression tasks will be automatically sorted and saved under distinct subdirectories.



Plotting

To plot the results, simply type “plot_plotData”.

We can define a specific metric to plot by using the “plotType” parameter. By default, it plots ROC curves, CPU time, prediction performance, and the mean true positive rates at certain false positive rates.

Certain FPR levels can also be customized using the “levels” parameters.

```
>> FPRlevels.levels.zeroZeroOne = 0.01;
FPRlevels.levels.zeroZeroFive = 0.05;
FPRlevels.levels.zeroOne = 0.1;
FPRlevels.levels

ans =

  struct with fields:

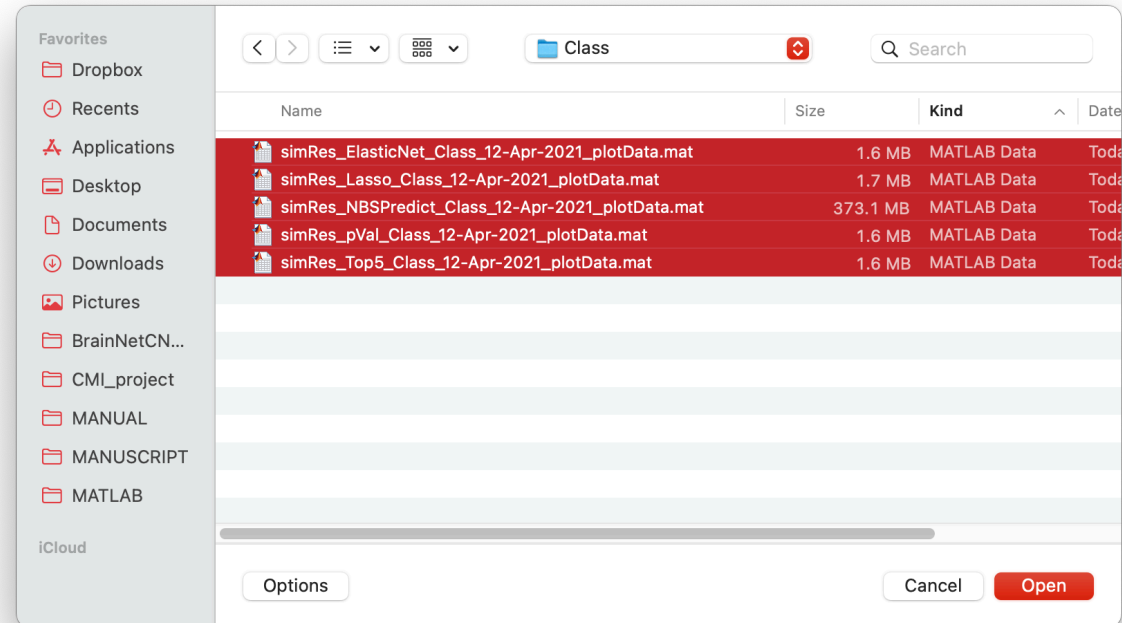
    zeroZeroOne: 0.0100
    zeroZeroFive: 0.0500
    zeroOne: 0.1000

>> plot_plotData('plotType',{ 'ROC', 'bar_TPR', 'predPerf', 'time'}, 'levels', FPRlevels);
```

Plotting

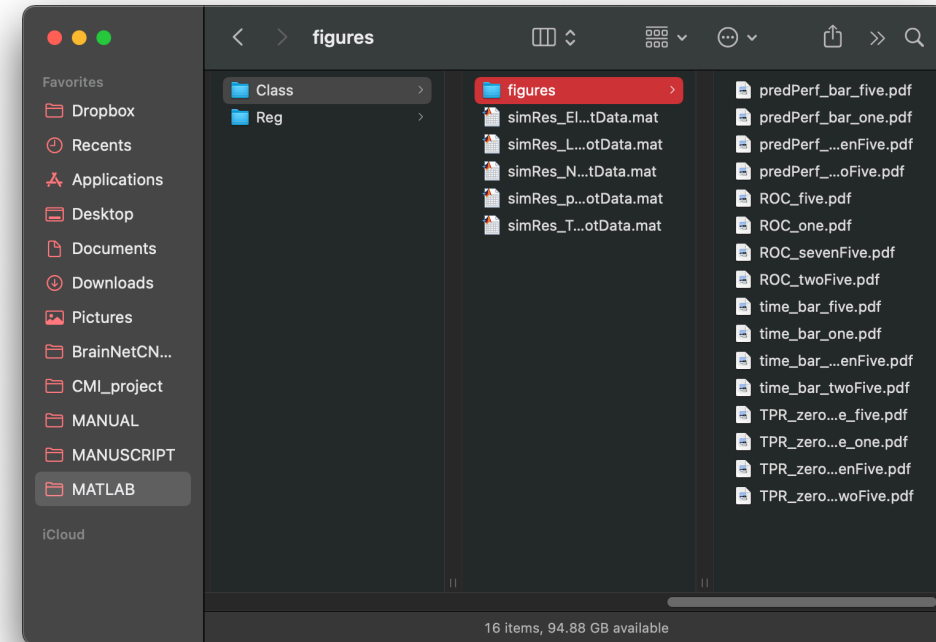
A window will pop up and ask to load processed results files.

Select all the processed files and load.



Plotting

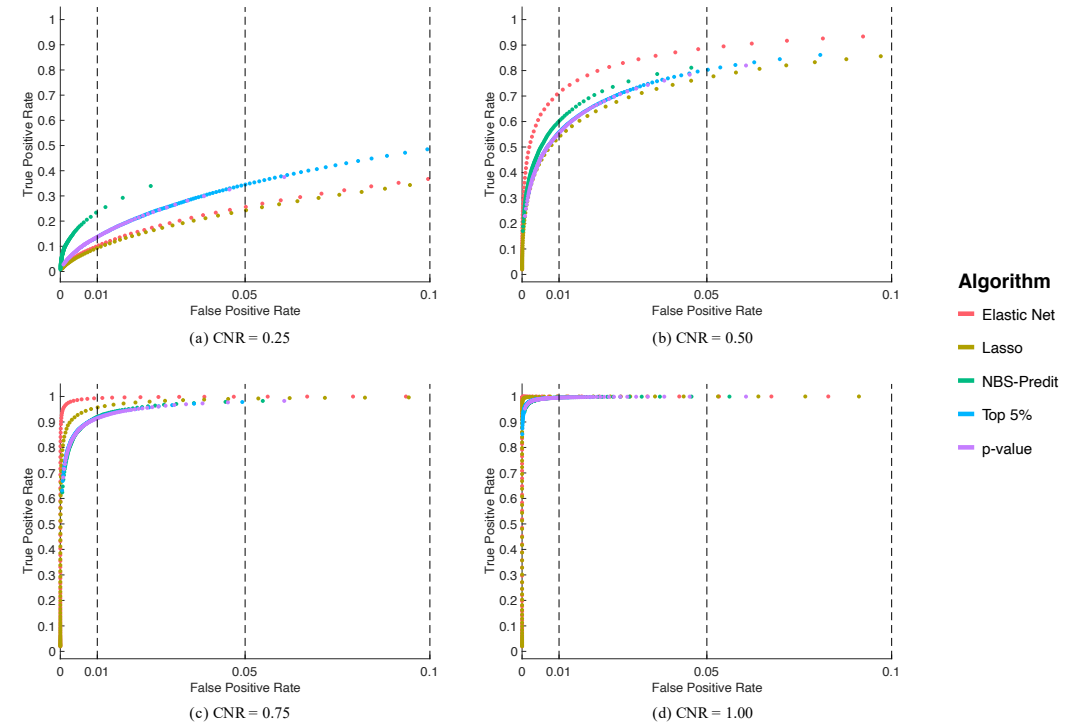
This function generates the figures and saves them in the “figures” folder.



Summary Results

On the first set of simulated data (i.e. classification), NBS-Predict outperformed the alternative algorithms in the 0.25 CNR condition and yielded comparable performance in other CNR conditions.

In the 0.50 and 0.75 CNR conditions, the elastic net algorithm was the best performing algorithm in identifying edges with contrast.

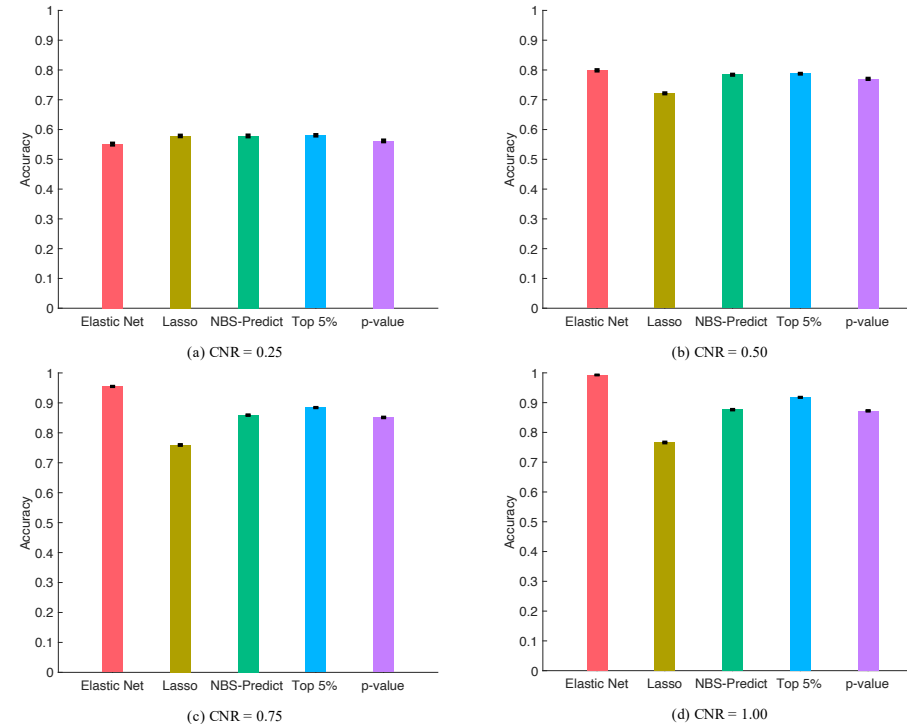


Receiver Operating Characteristic (ROC) curves were used to evaluate specificity and sensitivity of NBS-Predict and the alternative feature selection algorithms (Elastic net, Lasso, Top 5% of features, and p-value thresholding) in four different contrast-to-noise ratios conditions (CNR): (a) 0.25, (b) 0.50, (c) 0.75 and (d) 1.00.

Summary Results

In the 0.25 CNR condition, NBS-Predict outperformed the elastic net and p-value thresholding algorithms while yielding similar classification accuracy with the lasso and top 5%.

However, the elastic net performed the best in other CNR conditions.

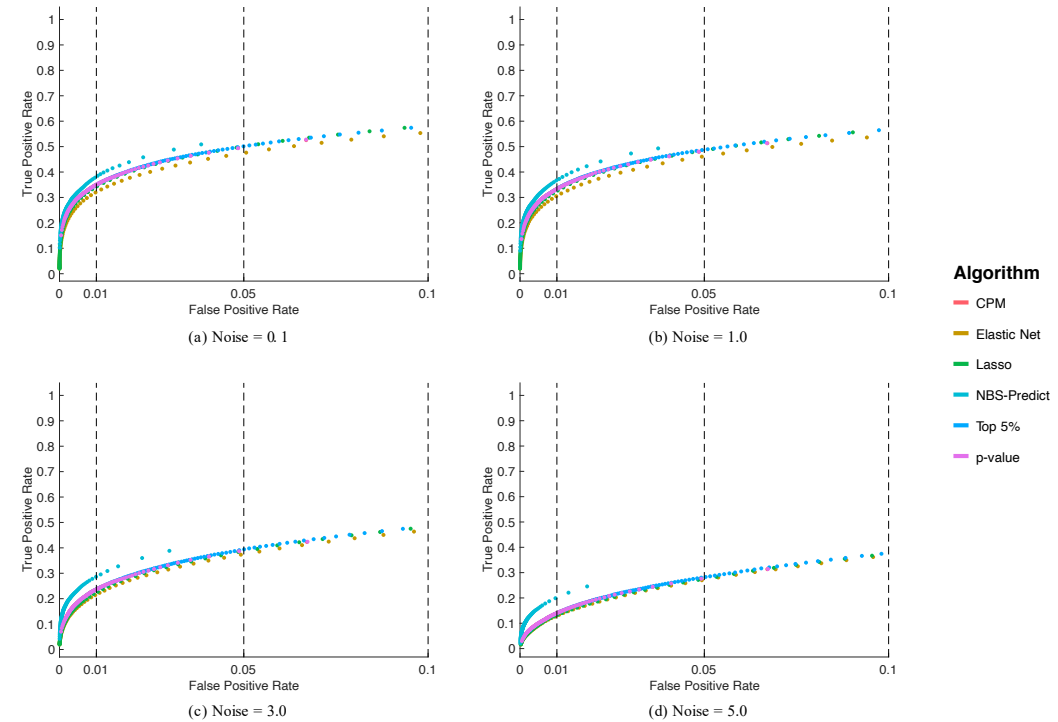


The classification accuracy of NBS-Predict and the alternative algorithms (elastic net, lasso, top 5% of features, and p-value thresholding) in four contrast-to-noise ratio (CNR) conditions: (a) 0.25, (b) 0.50, (c) 0.75 and (d) 1.00. Error bars depicted in each plot represents 95% confidence interval ($p < 0.05$).

Summary Results

On the second set of simulated data (i.e. regression), NBS-Predict outperformed all the alternative algorithms.

The advantage of NBS-Predict over the alternative algorithms increased with noise.

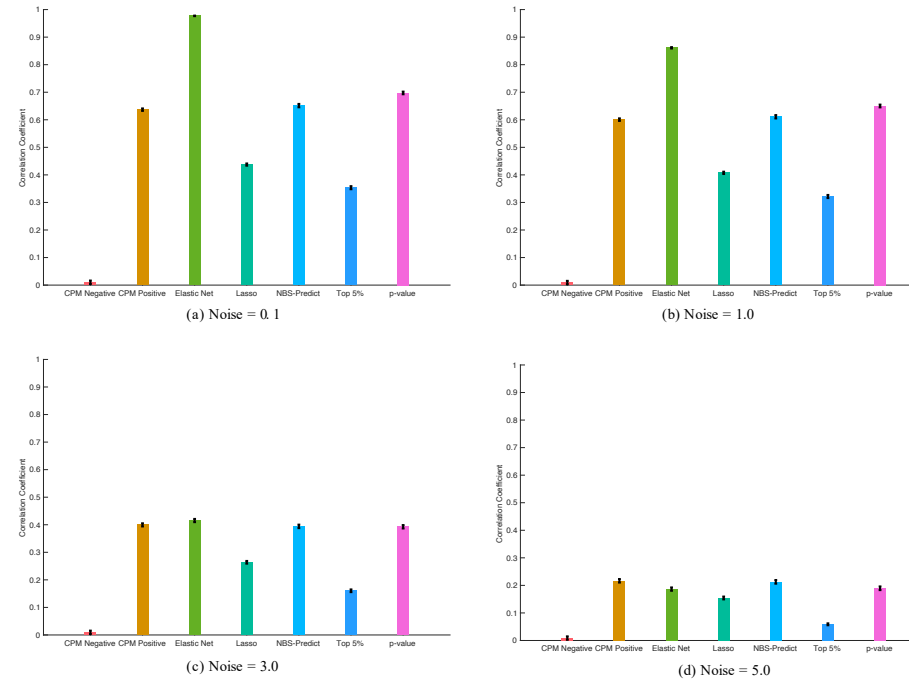


Receiver Operating Characteristic (ROC) curves were used to evaluate specificity and sensitivity of NBS-Predict and the alternative algorithms (Elastic net, Lasso, Top 5% of features, p-value thresholding feature selection algorithms, and CPM) in four different noise conditions: (a) 0.01, (b) 1.00, (c) 3.0, (d) 5.0.

Summary Results

Regarding target prediction, the elastic net outperformed other algorithms, including NBS-Predict, in most noise conditions, except for the “Noise = 5.0” condition.

NBS-Predict outperformed the top 5% and lasso algorithms in all conditions while yielding similar performance with the p-value thresholding and CPM with the sets of positive edges.



The performance (i.e. Pearson's correlation coefficient) of NBS-Predict and the alternative algorithms (elastic net, lasso, top 5% of features, and p-value thresholding) in predicting a target variable in four noise conditions (a) 0.1, (b) 1.0, (c) 3.0, (d) 5.0. Error bars depicted in each plot represents 95% confidence interval ($p < 0.05$).