minepy Documentation

Release 1.3.0

Davide Albanese

Feb 24, 2017

Quick start

minepy provides an **ANSI C library** for the Maximal Information-based Nonparametric Exploration (**MIC** and **MINE family**). Key features:

- **APPROX-MIC** (the original algorithm, DOI: 10.1126/science.1205438) and **MIC_e** (DOI: arXiv:1505.02213 and DOI: arXiv:1505.02214) estimators;
- **Total Information Coefficient** (TIC, DOI: arXiv:1505.02213) and the **Generalized Mean Information Coefficient** (GMIC, DOI: arXiv:1308.5712);
- a **C++** interface;
- an efficient **Python API**;
- an efficient **MATLAB/OCTAVE API**;
- a **command-line application** similar to the original `MINE.jar`;
- the *minerva* **R interface** is available at CRAN.

minepy is an open-source, GPLv3-licensed software.

- Homepage
- Download
- Github page
- Issues

CHAPTER 1

Documentation

- Latest (github snapshot)
- 1.2 (stable, 16/10/27)
- 1.1 (stable, 16/05/10)
- 1.0

CHAPTER 2

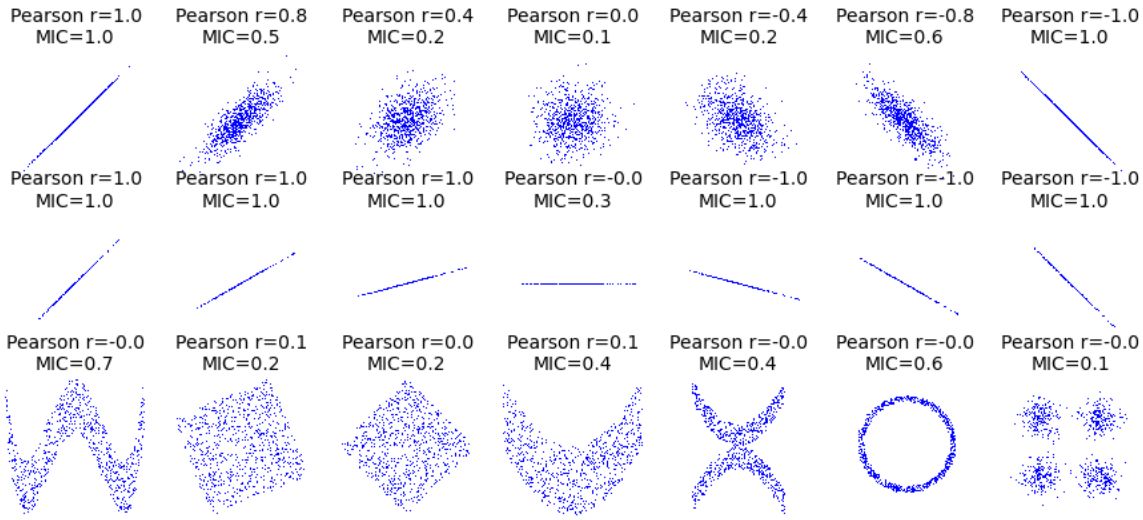
Citing minepy

Davide Albanese, Michele Filosi, Roberto Visintainer, Samantha Riccadonna, Giuseppe Jurman and Cesare Furlanello. minerva and minepy: a C engine for the MINE suite and its R, Python and MATLAB wrappers. Bioinformatics (2013) 29(3): 407-408 first published online December 14, 2012 doi:10.1093/bioinformatics/bts707.

CHAPTER 3

Financial Contributions

- Computational Biology Unit - Research and Innnovation Center at Fondazione Edmund Mach
- Predictive Models for Biological and Environmental Data Analysis (MPBA) Research Unit at Fondazione Bruno Kessler



Download and Install

C and C++ users

Requirements:

- GCC

Download latest version from https://github.com/minepy/minepy/releases. No installation is required. See *C API* and *C++ API* on how to include and compile the library in your C/C++ software.

Python users

Requirements:

- GCC
- Python >= 2.7
- NumPy >= 1.3.0 (with header files)

On Linux

We suggest to install the GCC compiler through the package manager (e.g. on Ubuntu/Debian):

```
sudo apt-get update
sudo apt-get install build-essential python-dev
```

Then, upgrade pip and install setuptools:

```
pip install --upgrade pip
pip install 'setuptools >=14.0'
```

Finally, install minepy:

```
sudo pip install minepy
```

On Mac OS X

In Mac OS X, we recommend to install Python from Homebrew:

1. Install Xcode;
2. Install Homebrew;
3. Make sure the environment variable `PATH` is properly setted in your `~/.bash_profile` or `~/.bashrc`:

   ```
   .. code-block:: sh
   ```

   export PATH=/usr/local/bin:$PATH

4. Install Python:

   ```
   brew update
   brew install python
   ```

Install gcc:

```
brew install gcc
```

Finally, install minepy:

```
sudo pip install minepy
```

Running the tests:

```
$ cd tests
$ python minepy_test.py
test_const (__main__.TestFunctions) ... ok
test_exp (__main__.TestFunctions) ... ok
test_linear (__main__.TestFunctions) ... ok
test_sine (__main__.TestFunctions) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.412s

OK
```

MATLAB users (Windows, Linux and Mac OS X)

The library works with MATLAB >= 7.3 (R2006b) since it supports 64-Bit indexing. See http://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-use-64-bit-api.html. Download latest version from https://github.com/minepy/minepy/releases.

1. Untar the file `minepy-X.Y.Z.tar.gz` (where `X.Y.Z` is the current version of minepy)
2. Open MATLAB
3. In the MATLAB "command window" go into the `minepy-X.Y.Z` folder by typing:

```
>> cd path_to_minepy-X.Y.Z/matlab/
```

4. Build the binary MEX file by typing:

```
>> mex mine_mex.c ../libmine/mine.c
```

5. Now you have the binary MEX-file in `path_to_minepy-X.Y.Z/matlab/` (`mine_mex.mex*`, where * can be `a64`, `maci64`, `w32` or `w64`)
6. Put your MEX-file (`mine_mex.mex*`) and `mine.m` in a folder on the MATLAB path. Alternatively, you can add `path_to_minepy-X.Y.Z/matlab/` selecting File > SetPath.
7. Test the MEX-file by typing:

```
>> minestats = mine([1,2,3,4,5,6], [1,2,3,4,5,6])
```

OCTAVE users (Windows, Linux and Mac OS X)

Download latest version from https://github.com/minepy/minepy/releases.

1. Untar the file `minepy-X.Y.Z.tar.gz` (where `X.Y.Z` is the current version of minepy)
2. Run OCTAVE
3. Go into the `minepy-X.Y.Z` folder by typing:

```
octave:1> cd path_to_minepy-X.Y.Z/matlab/
```

4. Build the binary MEX-file by typing:

```
octave:2> mex mine_mex.c ../libmine/mine.c
```

5. Now, you have the binary MEX-file in `path_to_minepy-X.Y.Z/matlab/` (`mine_mex.mex`)

6. Put your MEX-file (`mine_mex.mex`) and `mine.m` in a folder on the OCTAVE path. Alternatively, you can add `path_to_minepy-X.Y.Z/matlab/` in the OCTAVE startup file (http://www.gnu.org/software/octave/doc/interpreter/Startup-Files.html)
7. Test the MEX-file by typing:

```
octave:3> minestats = mine([1,2,3,4,5,6], [1,2,3,4,5,6])
```

MINE Application

The mine application is a script (installed together the minepy library) which computes the MINE statistics on a comma-separated values (CSV) file. The first column of the file must contain the variable names and each variable must have the same number of samples. The file must be in the form:

```
var1_name,1,2.5,3,4,5,6,7,8,9,10
var2_name,8,7,6,5,6,6,6,1.23,4,4
var3_name,1,7,3,5,6,6,6,3,4,4.2
var4_name,...
...
```

The input file can contain missing values (new in minepy 1.1.0):

```
var1_name,1,2.5,3,4,5,,7,8,9,10
var2_name,8,7,,5,6,6,6,1.23,4,4
var3_name,1,7,3,,,6,6,3,4,4.2
var4_name,...
...
```

Only the samples without missing values are used to compute the MINE statistics. For instance, var2_name vs. var3_name:

```
{8,7, ,5,6,6,6,1.23,4,4  } -> {8,7,6,6,1.23,4,4  }
{1,7,3, , ,6,6,3   ,4,4.2} -> {1,7,6,6,3   ,4,4.2}
```

Usage:

```
Usage: mine infile [-a <alpha>] [-c <c>] [-o <file>] [-m <var index>] [-p <var1 index>
↪ <var2 index>]

MINE Python v. 1.1.0 [Homepage: minepy.sf.net]. The mine script
compares by default all pairs of variables against each other. It
writes an output file where each column contains MIC (strength),
MIC-r^2 (nonlinearity), MAS (non- monotonicity), MEV (functionality),
MCN (complexity, eps=0), MCN_GENERAL (complexity, eps=1-MIC) and
Pearson (r). The input must be a comma-separated values file where the
first column must contain the variable names. Each variable must have
the same numer of samples.

Options:
  -h, --help            show this help message and exit
  -a <alpha>, --alpha=<alpha>
                        the exponent in B(n) = n^alpha (default: 0.6.) alpha
                        must be in (0, 1.0]
  -c <c>, --clumps=<c>  determines how many more clumps there will be than
                        columns in every partition. Default value is 15,
                        meaning that when trying to draw Gx grid lines on the
```

```
                        x-axis, the algorithm will start with at most 15*Gx
                        clumps (default: 15). c must be > 0
-o <file>, --output=<file>
                        output filename (default: mine_out.csv)
-m <var index>, --master=<var index>
                        variable <var index> vs. all <var index> must be in
                        [1, number of variables in file]
-p <var1 index> <var2 index>, --pair=<var1 index> <var2 index>
                        variable <var1 index> vs. variable <var2 index> <var1
                        index> and <var2 index> must be in [1, number of
                        variables in file]
```

Examples

Spellman Gene Expression dataset

Compute the MINE statistics for variable #1 (time) vs. all the other variables, with alpha=0.67 and c=15.

1. Download the Spellman dataset
2. From a terminal, run:

```
$ mine Spellman.csv -a 0.67 -c 15 -m 1 -o Spellman_MINE.txt
```

Dataset details are in http://www.exploredata.net/Downloads

Baseball dataset

Compute the MINE statistics all pairs of variables against each other, with alpha=0.7 and c=15.

1. Download the MLB2008 dataset
2. From a terminal, run:

```
$ mine MLB2008.csv -a 0.7 -c 15 -o MLB2008_MINE.txt
```

Dataset details are in http://www.exploredata.net/Downloads

Microbiome dataset

Compute the MINE statistics all pairs of variables against each other, with alpha=0.551 and c=10.

1. Download the Microbiome dataset
2. From a terminal, run:

```
$ mine Microbiome.csv -a 0.551 -c 10 -o Microbiome_MINE.txt
```

Dataset details are in http://www.exploredata.net/Downloads

C API

This chapter describes the mine C library. These functions and structures are declared in the header file `mine.h`, located in the `libmine/` folder. You need to add `#include "mine.h"` in your C source files and link your program with `mine.c`.

Defines

EST_MIC_APPROX **0**
: Original estimator described in DOI: 10.1126/science.1205438.

EST_MIC_E **1**
: Estimator described in DOI: arXiv:1505.02213 and DOI: arXiv:1505.02214.

FALSE **0**
: False value.

TRUE **1**
: True value.

char ***libmine_version**
: The libmine version in the form X.Y.Z (e.g., 1.0.1).

Structures

mine_problem
: The mine_problem structure describes the problem. x and y are the two variables of length n.

```
typedef struct mine_problem
  {
    int n;
    double *x;
    double *y;
  } mine_problem
```

mine_parameter
: MINE parameters.

alpha [(0, 1.0] or >=4] if alpha is in (0,1] then B will be max(n^alpha, 4) where n is the number of samples. If alpha is >=4 then alpha defines directly the B parameter. If alpha is higher than the number of samples (n) it will be limited to be n, so B = min(alpha, n).

c [(> 0)] determines how many more clumps there will be than columns in every partition. Default value is 15, meaning that when trying to draw x grid lines on the x-axis, the algorithm will start with at most 15*x clumps.

est [(EST_MIC_APPROX, EST_MIC_E)] estimator. With est=EST_MIC_APPROX the original MINE statistics will be computed, with est=EST_MIC_E the equicharacteristic matrix is is evaluated and the mine_mic() and mine_tic() functions will return MIC_e and TIC_e values respectively.

```
typedef struct mine_parameter
  {
    double alpha;
    double c;
    int est;
  } mine_parameter
```

mine_score
The mine_score structure contains the maximum normalized mutual information scores (i.e. the characteristic matrix if est=EST_MIC_APPROX, the equicharacteristic matrix instead). M[i][j] contains the score using a grid partitioning x-values into i+2 bins and y-values into j+2 bins. m and M are of length n and each M[i] is of length m[i].

```
typedef struct mine_score
  {
    int n;       /* number of rows of M */
    int *m;      /* number of cols of M[i] for each i */
    double **M; /* the (equi)characteristic matrix */
  } mine_score
```

Functions

mine_score ***mine_compute_score** (*mine_problem *prob*, *mine_parameter *param*)
Computes the (equi)characteristic matrix (i.e. maximum normalized mutual information scores) and returns a mine_score structure. Returns NULL if an error occurs.

char ***mine_check_parameter** (*mine_parameter *param*)
Checks the parameters. This function should be called before calling mine_compute_score(). It returns NULL if the parameters are feasible, otherwise an error message is returned. See the mine_parameter documentation.

double **mine_mic** (*mine_score *score*)
Returns the Maximal Information Coefficient (MIC or MIC_e).

double **mine_mas** (*mine_score *score*)
Returns the Maximum Asymmetry Score (MAS).

double **mine_mev** (*mine_score *score*)
Returns the Maximum Edge Value (MEV).

double **mine_mcn** (*mine_score *score*, double *eps*)
Returns the Minimum Cell Number (MCN) with epsilon >= 0.

double **mine_mcn_general** (*mine_score *score*)
Returns the Minimum Cell Number (MCN) with eps=1 - MIC.

double **mine_tic** (*mine_score *score*, int *norm*)
Returns the Total Information Coefficient (TIC or TIC_e). norm=TRUE normalizes the Total Information Coefficient, returning values in [0, 1].

double **mine_gmic** (*mine_score *score*)
Returns the Generalized Mean Information Coefficient (GMIC).

void **mine_free_score** (*mine_score **score*)
This function frees the memory used by a mine_score and destroys the score structure.

Convenience structures and functions

mine_matrix
Mine matrix, variables x samples.

```
typedef struct mine_matrix
  {
    double *data; /* matrix in row-major order */
    int n;        /* number of rows */
```

```
    int m;         /* number of cols */
  } mine_matrix
```

mine_pstats

For each statistic, the upper triangle of the matrix is stored by row (condensed matrix). If m is the number of variables, then for i<j<m, the statistic between (row) i and j is stored in k = m*i - i*(i+1)/2 - i - 1 + j. The length of the vectors is n = m*(m-1)/2.

```
typedef struct mine_pstats
  {
    double *mic; /* condensed matrix */
    double *tic; /* condensed matrix */
    int n;       /* number of elements */
  } mine_pstats
```

mine_cstats

For each statistic, the matrix is stored by row. If n and m are the number of variables in X and Y respectively, then the statistic between the (row) i (for X) and j (for Y) is stored in k = i*m + j. The length of the vector is d = n*m.

```
typedef struct mine_cstats
  {
    double *mic; /* matrix in row-major order */
    double *tic; /* matrix in row-major order */
    int n;       /* number of rows */
    int m;       /* number of cols */
  } mine_cstats
```

mine_pstats ***mine_compute_pstats**(*mine_matrix *X*, *mine_parameter *param*)

Compute pairwise statistics (MIC and normalized TIC) between variables.

mine_cstats ***mine_compute_cstats**(*mine_matrix *X*, *mine_matrix *Y*, *mine_parameter *param*)

Compute statistics (MIC and normalized TIC) between each pair of the two collections of variables.

Example

The example is located in `examples/c_example.c`.

```
/*
 * $ gcc c_example.c -O3 -Wall ../libmine/mine.c -I../libmine/ -lm
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "mine.h"


int main (int argc, char **argv)
{
  mine_problem prob;
  mine_parameter param;
  mine_score *score;

  double PI = 3.14159265;
```

```
int i, j;
char *ret;

printf("libmine version %s\n", libmine_version);

/* set the parameters */
param.alpha = 0.6;
param.c = 15;
param.est = EST_MIC_APPROX;

/* check the parameters */
ret = mine_check_parameter(&param);
if (ret)
  {
    printf("ERROR: %s\n\n", ret);
    return 1;
  }

/* build the problem */
prob.n = 201;
prob.x = (double *) malloc (prob.n * sizeof (double));
prob.y = (double *) malloc (prob.n * sizeof (double));
for (i=0; i<prob.n; i++)
  {
    /* build x = [0, 0.005, ..., 1] */
    prob.x[i] = (double) i / (double) (prob.n-1);

    /* build y = sin(10 * pi * x) + x */
    prob.y[i] = sin(10 * PI * prob.x[i]) + prob.x[i];
  }

/* compute score */
clock_t start = clock();
score = mine_compute_score(&prob, &param);
clock_t end = clock();
printf("Elapsed time: %.6f seconds\n\n",
       (double)(end - start) / CLOCKS_PER_SEC);

if (score == NULL)
  {
    printf("ERROR: mine_compute_score()\n");
    return 1;
  }

/* print some MINE statistics */
printf ("Some MINE statistics:\n\n");
printf ("MIC: %.3lf\n", mine_mic(score));
printf ("MEV: %.3lf\n", mine_mev(score));
printf ("MCN (eps=0): %.3lf\n", mine_mcn(score, 0));


/* print the characteristic matrix M */
printf ("\nCharacteristic Matrix:\n\n");

for (i=0; i<score->n; i++)
  {
    for (j=0; j<score->m[i]; j++)
            printf ("%.3lf ", score->M[i][j]);
```

```
      printf ("\n");
    }

  /* free score */
  mine_free_score(&score);

  /* free prob */
  free(prob.x);
  free(prob.y);

  return 0;
}
```

To compile the example, open a terminal, go into the example (`examples/`) folder and run:

```
$ gcc c_example.c -Wall ../libmine/mine.c -I../libmine/ -lm
```

or

```
$ gcc c_example.c -O3 -Wall ../libmine/mine.c -I../libmine/ -lm
```

for an extensive optimization. Run the example by typing:

```
$ ./a.out
libmine version 1.1.0
Elapsed time: 0.011288 seconds

Some MINE statistics:

MIC: 1.000
MEV: 1.000
MCN (eps=0): 4.585

Characteristic Matrix:

0.108 0.146 0.226 0.347 0.434 0.545 0.639 0.740 0.863 0.932 1.000
0.199 0.138 0.169 0.256 0.298 0.379 0.427
0.237 0.190 0.217 0.286 0.324
0.247 0.198 0.191
0.262 0.213 0.232
0.272 0.225
0.286 0.237
0.296
0.308
0.321
0.333
```

Example (convenience functions)

The example is located in `examples/c_conv_example.c`.

```
/*
 * $ gcc c_conv_example.c -O3 -Wall ../libmine/mine.c -I../libmine/ -lm
 */

#include <stdlib.h>
```

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "mine.h"


int main (int argc, char **argv)
{
  mine_parameter param;
  mine_matrix X, Y;
  mine_pstats *pstats;
  mine_cstats *cstats;

  int i, j, k, z;
  char *ret;


  printf("libmine version %s\n\n", libmine_version);

  /* set the parameters */
  param.alpha = 9;
  param.c = 5;
  param.est = EST_MIC_E;

  /* check the parameters */
  ret = mine_check_parameter(&param);
  if (ret)
    {
      printf("ERROR: %s\n", ret);
      return 1;
    }

  /* build the X matrix */
  X.n = 8;   /* 8 variables */
  X.m = 320; /* 320 samples */
  X.data = (double *) malloc ((X.n * X.m) * sizeof(double));
  for (i=0; i<(X.n * X.m); i++)
    X.data[i] = (double) rand() / (RAND_MAX);

  /* build the Y matrix */
  Y.n = 4;    /* 4 variables */
  Y.m = 320;  /* 320 samples */
  Y.data = (double *) malloc ((Y.n * Y.m) * sizeof(double));
  for (i=0; i<(Y.n * Y.m); i++)
    Y.data[i] = (double) rand() / (RAND_MAX);

  /* compute pairwise statistics between samples in X */
  pstats = mine_compute_pstats(&X, &param);

  if (pstats == NULL)
    {
      printf("ERROR: pstats()\n");
      return 1;
    }

  /* print the TIC matrix */
  printf("     ");
  for (j=1; j<X.n; j++)
```

```
    printf("X[%d]  ", j);
  printf("\n");
  k = 0;
  for (i=0; i<X.n-1; i++)
    {
      printf("X[%d] ", i);
      for (z=0; z<i; z++)
        printf("      ");

      for (j=i+1; j<X.n; j++)
        {
          printf("%.3lf ", pstats->tic[k]);
          k++;
        }
      printf("\n");
    }

  /* free pstats */
  free(pstats->mic);
  free(pstats->tic);
  free(pstats);

  printf("\n");

  /* compute statistics between each pair of samples in X and Y */
  cstats = mine_compute_cstats(&X, &Y, &param);

  if (cstats == NULL)
    {
      printf("ERROR: cstats()\n");
      return 1;
    }

  /* print the TIC matrix */
  printf("     ");
  for (j=0; j<Y.n; j++)
    printf("Y[%d]  ", j);
  printf("\n");
  for (i=0; i<X.n; i++)
    {
      printf("X[%d] ", i);
      for (j=0; j<Y.n; j++)
        printf("%.3lf ", cstats->tic[(i*Y.n)+j]);
      printf("\n");
    }

  /* free cstats */
  free(cstats->mic);
  free(cstats->tic);
  free(cstats);

  /* free data */
  free(X.data);
  free(Y.data);

  return 0;
}
```

To compile the example, open a terminal, go into the example (`examples/`) folder and run:

```
$ gcc c_conv_example.c -O3 -Wall ../libmine/mine.c -I../libmine/ -lm
```

Run the example by typing:

```
$ ./a.out
libmine version 1.2.0

     X[1]  X[2]  X[3]  X[4]  X[5]  X[6]  X[7]
X[0] 0.008 0.011 0.011 0.013 0.011 0.007 0.010
X[1]       0.010 0.012 0.015 0.020 0.010 0.011
X[2]             0.015 0.016 0.013 0.012 0.019
X[3]                   0.016 0.016 0.012 0.017
X[4]                         0.010 0.013 0.016
X[5]                               0.010 0.015
X[6]                                     0.018

     Y[0]  Y[1]  Y[2]  Y[3]
X[0] 0.012 0.012 0.016 0.010
X[1] 0.010 0.011 0.013 0.014
X[2] 0.013 0.011 0.012 0.016
X[3] 0.016 0.009 0.014 0.011
X[4] 0.007 0.022 0.018 0.017
X[5] 0.011 0.015 0.015 0.009
X[6] 0.012 0.007 0.015 0.013
X[7] 0.017 0.010 0.010 0.009
```

C++ API

This chapter describes the mine C++ wrapper. The class is declared in the header file `cppmine.h`, located in the `libmine/` folder. You need to add `#include "cppmine.h"` in your C++ source files and link your program with `mine.c` and `cppmine.c`.

See the *C API* documentation.

class MINE

MINE::**MINE**(double *alpha*, double *c*, int *est*)
Constructor. Throws an exception when the parameters are invalid.

MINE::**~MINE**()
Destructor.

void MINE::**compute_score**(double *x, double *y, int *n*)

void MINE::**mic**()

void MINE::**mas**()

void MINE::**mev**()

void MINE::**mcn**(double *eps*)

void MINE::**mcn_general**()

void MINE::**tic**(int *norm*)

Example

The example is located in `examples/cpp_example.cpp`.

```
#include <cstdlib>
#include <cmath>
#include <iostream>
#include "cppmine.h"

using namespace std;

int
main (int argc, char **argv)
{
  double PI;
  int i, n;
  double *x, *y;
  MINE *mine;

  PI = 3.14159265;

  /* build the MINE object with exceptions management */
  try
    {
      mine = new MINE(0.6, 15, EST_MIC_APPROX);
    }
  catch (char *s)
    {
      cout << "WARNING: " << s << "\n";
      cout << "MINE will be set with alpha=0.6 and c=15" << "\n";
      mine = new MINE(0.6, 15, EST_MIC_APPROX);
    }

  /* build the problem */
  n = 1001;
  x = new double [n];
  y = new double [n];
  for (i=0; i<n; i++)
    {
      /* build x = [0, 0.001, ..., 1] */
      x[i] = (double) i / (double) (n-1);

      /* build y = sin(10 * pi * x) + x */
      y[i] = sin(10 * PI * x[i]) + x[i];
    }

  /* compute score with exceptions management */
  try
    {
      mine->compute_score(x, y, n);
    }
  catch (char *s)
    {
      cout << "ERROR: " << s << "\n";
      return 1;
    }

  /* print mine statistics */
```

```
  try
    {
      cout << "MIC: " << mine->mic() << "\n";
      cout << "MAS: " << mine->mas() << "\n";
      cout << "MEV: " << mine->mev() << "\n";
      cout << "MCN (eps=0): " << mine->mcn(0) << "\n";
      cout << "MCN (eps=1-MIC): " << mine->mcn_general() << "\n";
      cout << "TIC: " << mine->tic(FALSE) << "\n";
    }
  catch (char *s)
    {
      cout << "ERROR: " << s << "\n";
      return 1;
    }

  /* delete the mine object */
  delete mine;

  /* free the problem */
  delete [] x;
  delete [] y;

  return 0;
}
```

To compile the example, open a terminal, go into the example (`examples/`) folder and run:

```
$ g++ -O3 -Wall -Wno-write-strings cpp_example.cpp ../libmine/cppmine.cpp \
  ../libmine/mine.c -I../libmine/
```

Run the example by typing:

```
MIC: 0.999999
MAS: 0.728144
MEV: 0.999999
MCN (eps=0): 4.58496
MCN (eps=1-MIC): 4.58496
TIC: 67.5236
```

A more simple example

The example is located in `examples/cpp_example2.cpp`.

```
#include <cstdlib>
#include <cmath>
#include <iostream>
#include "cppmine.h"

using namespace std;

int
main (int argc, char **argv)
{
  int n = 7;
  double x[] = {1.,2.,3.,4.,5.,6.,7.};
  double y[] = {1.,2.,3.,4.,3.,2.,1.};
```

```
  /* build the MINE object */
  MINE mine(0.6, 15, EST_MIC_APPROX);

  /* compute score */
  mine.compute_score(x, y, n);

  /* print MIC */
  cout << "MIC: " << mine.mic() << "\n";

  return 0;
}
```

To compile the example, open a terminal, go into the example (`examples/`) folder and run:

```
$ g++ -O3 -Wall -Wno-write-strings cpp_example2.cpp ../libmine/cppmine.cpp \
  ../libmine/mine.c -I../libmine/
```

Run the example by typing:

```
$ ./a.out
MIC: 0.291692
```

Python API

class `minepy.`**MINE**(*alpha=0.6*, *c=15*, *est="mic_approx"*)

Maximal Information-based Nonparametric Exploration.

Parameters

- **alpha** (*float (0, 1.0] or >=4*) – if alpha is in (0,1] then B will be max(n^alpha, 4) where n is the number of samples. If alpha is >=4 then alpha defines directly the B parameter. If alpha is higher than the number of samples (n) it will be limited to be n, so B = min(alpha, n).
- **c** (*float (> 0)*) – determines how many more clumps there will be than columns in every partition. Default value is 15, meaning that when trying to draw x grid lines on the x-axis, the algorithm will start with at most 15*x clumps.
- **est** (*str ("mic_approx", "mic_e")*) – estimator. With est="mic_approx" the original MINE statistics will be computed, with est="mic_e" the equicharacteristic matrix is is evaluated and the mic() and tic() methods will return MIC_e and TIC_e values respectively.

compute_score(*x*, *y*)

Computes the (equi)characteristic matrix (i.e. maximum normalized mutual information scores.

mic()

Returns the Maximal Information Coefficient (MIC or MIC_e).

mas()

Returns the Maximum Asymmetry Score (MAS).

mev()

Returns the Maximum Edge Value (MEV).

mcn(*eps=0*)
Returns the Minimum Cell Number (MCN) with eps >= 0.

mcn_general()
Returns the Minimum Cell Number (MCN) with eps = 1 - MIC.

gmic(*p=-1*)
Returns the Generalized Maximal Information Coefficient (GMIC).

tic(*norm=False*)
Returns the Total Information Coefficient (TIC or TIC_e). If norm==True TIC will be normalized in [0, 1].

get_score()
Returns the maximum normalized mutual information scores (i.e. the characteristic matrix M if est="mic_approx", the equicharacteristic matrix instead). M is a list of 1d numpy arrays where M[i][j] contains the score using a grid partitioning x-values into i+2 bins and y-values into j+2 bins.

computed()
Return True if the (equi)characteristic matrix) is computed.

Convenience functions

minepy.**pstats**(*X*, *alpha=0.6*, *c=15*, *est="mic_approx"*)
Compute pairwise statistics (MIC and normalized TIC) between variables (convenience function).

For each statistic, the upper triangle of the matrix is stored by row (condensed matrix). If m is the number of variables, then for i < j < m, the statistic between (row) i and j is stored in k = m*i - i*(i+1)/2 - i - 1 + j. The length of the vectors is n = m*(m-1)/2.

Parameters

- **X** (*2D array_like object*) – An n-by-m array of n variables and m samples.
- **alpha** (*float (0, 1.0] or >=4*) – if alpha is in (0,1] then B will be max(n^alpha, 4) where n is the number of samples. If alpha is >=4 then alpha defines directly the B parameter. If alpha is higher than the number of samples (n) it will be limited to be n, so B = min(alpha, n).
- **c** (*float (> 0)*) – determines how many more clumps there will be than columns in every partition. Default value is 15, meaning that when trying to draw x grid lines on the x-axis, the algorithm will start with at most 15*x clumps.
- **est** (*str ("mic_approx", "mic_e")*) – estimator. With est="mic_approx" the original MINE statistics will be computed, with est="mic_e" the equicharacteristic matrix is is evaluated and MIC_e and TIC_e are returned.

Returns

- **mic** (*1D ndarray*) – the condensed MIC statistic matrix of length n*(n-1)/2.
- **tic** (*1D ndarray*) – the condensed normalized TIC statistic matrix of length n*(n-1)/2.

minepy.**cstats**(*X*, *Y*, *alpha=0.6*, *c=15*, *est="mic_approx"*)
Compute statistics (MIC and normalized TIC) between each pair of the two collections of variables (convenience function).

If n and m are the number of variables in X and Y respectively, then the statistic between the (row) i (for X) and j (for Y) is stored in mic[i, j] and tic[i, j].

Parameters

- **X** (*2D array_like object*) – An n by m array of n variables and m samples.
- **Y** (*2D array_like object*) – An p by m array of p variables and m samples.
- **alpha** (*float (0, 1.0] or >=4*) – if alpha is in (0,1] then B will be max(n^alpha, 4) where n is the number of samples. If alpha is >=4 then alpha defines directly the B parameter. If alpha is higher than the number of samples (n) it will be limited to be n, so B = min(alpha, n).
- **c** (*float (> 0)*) – determines how many more clumps there will be than columns in every partition. Default value is 15, meaning that when trying to draw x grid lines on the x-axis, the algorithm will start with at most 15*x clumps.
- **est** (*str ("mic_approx", "mic_e")*) – estimator. With est="mic_approx" the original MINE statistics will be computed, with est="mic_e" the equicharacteristic matrix is is evaluated and MIC_e and TIC_e are returned.

Returns

- **mic** (*2D ndarray*) – the MIC statistic matrix (n x p).
- **tic** (*2D ndarray*) – the normalized TIC statistic matrix (n x p).

First Example

The example is located in `examples/python_example.py`.

```
import numpy as np
from minepy import MINE

def print_stats(mine):
    print "MIC", mine.mic()
    print "MAS", mine.mas()
    print "MEV", mine.mev()
    print "MCN (eps=0)", mine.mcn(0)
    print "MCN (eps=1-MIC)", mine.mcn_general()
    print "GMIC", mine.gmic()
    print "TIC", mine.tic()

x = np.linspace(0, 1, 1000)
y = np.sin(10 * np.pi * x) + x
mine = MINE(alpha=0.6, c=15, est="mic_approx")
mine.compute_score(x, y)

print "Without noise:"
print_stats(mine)
print

np.random.seed(0)
y +=np.random.uniform(-1, 1, x.shape[0]) # add some noise
mine.compute_score(x, y)

print "With noise:"
print_stats(mine)
```

Run the example:

```
$ python python_example.py
Without noise:
```

```
MIC 1.0
MAS 0.726071574374
MEV 1.0
MCN (eps=0) 4.58496250072
MCN (eps=1-MIC) 4.58496250072
GMIC 0.779360251901
TIC 67.6612295532

With noise:
MIC 0.505716693417
MAS 0.365399904262
MEV 0.505716693417
MCN (eps=0) 5.95419631039
MCN (eps=1-MIC) 3.80735492206
GMIC 0.359475501353
TIC 28.7498326953
```

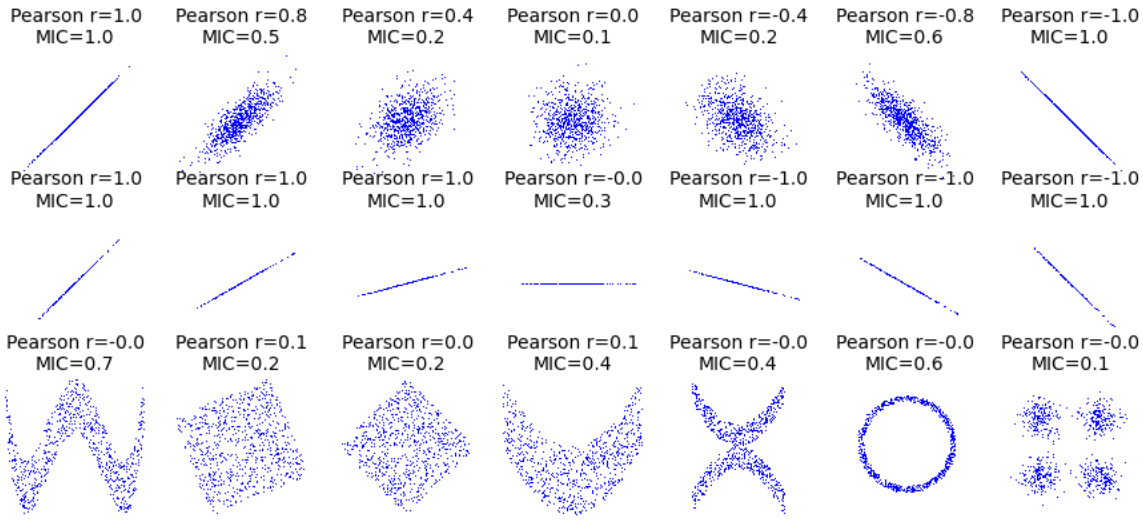
Second Example

The example is located in `examples/relationships.py`.

> **Warning:** Requires the matplotlib library.

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from minepy import MINE


rs = np.random.RandomState(seed=0)

def mysubplot(x, y, numRows, numCols, plotNum,
              xlim=(-4, 4), ylim=(-4, 4)):

    r = np.around(np.corrcoef(x, y)[0, 1], 1)
    mine = MINE(alpha=0.6, c=15, est="mic_approx")
    mine.compute_score(x, y)
    mic = np.around(mine.mic(), 1)
    ax = plt.subplot(numRows, numCols, plotNum,
                     xlim=xlim, ylim=ylim)
    ax.set_title('Pearson r=%.1f\nMIC=%.1f' % (r, mic),fontsize=10)
    ax.set_frame_on(False)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    ax.plot(x, y, ',')
    ax.set_xticks([])
    ax.set_yticks([])
    return ax

def rotation(xy, t):
    return np.dot(xy, [[np.cos(t), -np.sin(t)], [np.sin(t), np.cos(t)]])

def mvnormal(n=1000):
    cors = [1.0, 0.8, 0.4, 0.0, -0.4, -0.8, -1.0]
```

```
    for i, cor in enumerate(cors):
        cov = [[1, cor],[cor, 1]]
        xy = rs.multivariate_normal([0, 0], cov, n)
        mysubplot(xy[:, 0], xy[:, 1], 3, 7, i+1)

def rotnormal(n=1000):
    ts = [0, np.pi/12, np.pi/6, np.pi/4, np.pi/2-np.pi/6,
          np.pi/2-np.pi/12, np.pi/2]
    cov = [[1, 1],[1, 1]]
    xy = rs.multivariate_normal([0, 0], cov, n)
    for i, t in enumerate(ts):
        xy_r = rotation(xy, t)
        mysubplot(xy_r[:, 0], xy_r[:, 1], 3, 7, i+8)

def others(n=1000):
    x = rs.uniform(-1, 1, n)
    y = 4*(x**2-0.5)**2 + rs.uniform(-1, 1, n)/3
    mysubplot(x, y, 3, 7, 15, (-1, 1), (-1/3, 1+1/3))

    y = rs.uniform(-1, 1, n)
    xy = np.concatenate((x.reshape(-1, 1), y.reshape(-1, 1)), axis=1)
    xy = rotation(xy, -np.pi/8)
    lim = np.sqrt(2+np.sqrt(2)) / np.sqrt(2)
    mysubplot(xy[:, 0], xy[:, 1], 3, 7, 16, (-lim, lim), (-lim, lim))

    xy = rotation(xy, -np.pi/8)
    lim = np.sqrt(2)
    mysubplot(xy[:, 0], xy[:, 1], 3, 7, 17, (-lim, lim), (-lim, lim))

    y = 2*x**2 + rs.uniform(-1, 1, n)
    mysubplot(x, y, 3, 7, 18, (-1, 1), (-1, 3))

    y = (x**2 + rs.uniform(0, 0.5, n)) * \
        np.array([-1, 1])[rs.random_integers(0, 1, size=n)]
    mysubplot(x, y, 3, 7, 19, (-1.5, 1.5), (-1.5, 1.5))

    y = np.cos(x * np.pi) + rs.uniform(0, 1/8, n)
    x = np.sin(x * np.pi) + rs.uniform(0, 1/8, n)
    mysubplot(x, y, 3, 7, 20, (-1.5, 1.5), (-1.5, 1.5))

    xy1 = np.random.multivariate_normal([3, 3], [[1, 0], [0, 1]], int(n/4))
    xy2 = np.random.multivariate_normal([-3, 3], [[1, 0], [0, 1]], int(n/4))
    xy3 = np.random.multivariate_normal([-3, -3], [[1, 0], [0, 1]], int(n/4))
    xy4 = np.random.multivariate_normal([3, -3], [[1, 0], [0, 1]], int(n/4))
    xy = np.concatenate((xy1, xy2, xy3, xy4), axis=0)
    mysubplot(xy[:, 0], xy[:, 1], 3, 7, 21, (-7, 7), (-7, 7))

plt.figure(facecolor='white')
mvnormal(n=800)
rotnormal(n=200)
others(n=800)
plt.tight_layout()
plt.show()
```



Convenience functions example

The example is located in `examples/python_conv_example.py`.

```
import numpy as np
from minepy import pstats, cstats
import time

np.random.seed(0)

# build the X matrix, 8 variables, 320 samples
X = np.random.rand(8, 320)

# build the Y matrix, 4 variables, 320 samples
Y = np.random.rand(4, 320)

# compute pairwise statistics MIC_e and normalized TIC_e between samples in X,
# B=9, c=5
mic_p, tic_p =  pstats(X, alpha=9, c=5, est="mic_e")

# compute statistics between each pair of samples in X and Y
mic_c, tic_c =  cstats(X, Y, alpha=9, c=5, est="mic_e")

print "normalized TIC_e (X):"
print tic_p
print "MIC_e (X vs. Y):"
print mic_c
```

```
$ python python_conv_example.py
normalized TIC_e (X):
[ 0.01517556  0.00859132  0.00562575  0.01082706  0.01367201  0.0196697
  0.00947777  0.01273158  0.011291    0.01455822  0.0072817   0.01187837
  0.01595135  0.00902464  0.00974791  0.00952264  0.01806944  0.01064587
  0.00808622  0.01075486  0.00943122  0.01116569  0.01380142  0.01590193
  0.02159243  0.01450488  0.01347701  0.01036625]
MIC_e (X vs. Y):
[[ 0.0175473   0.01102385  0.01489008  0.02957048]
```

```
 [ 0.01294067  0.02682975  0.02743612  0.02224291]
 [ 0.01613576  0.0175808   0.01633154  0.02633199]
 [ 0.02090252  0.01680651  0.01735732  0.02186021]
 [ 0.01350926  0.01002233  0.02128154  0.02036634]
 [ 0.01459962  0.020248    0.0319421   0.01782455]
 [ 0.01186273  0.0291112   0.01577821  0.01970322]
 [ 0.012531    0.02071883  0.01536824  0.03312674]]
```

MATLAB and OCTAVE API

```
function [minestats, M] = mine(x, y, alpha, c, est)
% MINE  Maximal Information-based Nonparametric Exploration
%
%   Returns a struct containing MIC, MAS, MEV, MCN (eps=0) MCN_GENERAL
%   (eps=1-MIC) and TIC.
%
%   MINESTATS = MINE(X, Y, ALPHA, C, EST) computes the MINE statistics
%   between X and Y. X and Y must be row vectors of size n.
%   Alpha is the exponent in B(n) = n^alpha and must be in (0, 1.0].
%   Parameter c determines how many more clumps there will be than
%   columns in every partition and must be > 0.
%   Est is a string defining the estimation method.
%
%   MINESTATS = MINE(X, Y, ALPHA, C) computes the MINE statistics
%   between X and Y. Default value of EST is 'mic_approx'.
%
%   MINESTATS = MINE(X, Y, ALPHA) computes the MINE statistics
%   between X and Y. Default value of c is 15.
%
%   MINESTATS = MINE(X, Y) computes the MINE statistics
%   between X and Y. Default value of alpha is 0.6 and default value
%   of c is 15.
```

Example

The example is located in `examples/matlab_example.m`.

```
% create x = [0, 0.001, 0.002, ..., 0.998, 0.999, 1]
x = linspace(0, 1, 1001);
% y = sin(10 * pi * x) + x
y = sin(10 * pi * x) + x;
% compute the mine statistics
minestats = mine(x, y);
```

```
% print the minestats structure
minestats

minestats =

        mic: 1
        mas: 0.7261
        mev: 1
        mcn: 4.5850
```

```
mcn_general: 4.5850
        tic: 67.661
```

APPROX-MIC Implementation Details

The core implementation of libmine is built from scratch in ANSI C starting from the pseudocode provided in DOI: 10.1126/science.1205438, Supplementary On-line Material (SOM), as no original Java source code is available. The level of detail of the pseudocode leaves a few ambiguities and in this section we list and comment the most crucial choices we adopted for the algorithm steps whenever no explicit description was provided. Obviously, our choices are not necessarily the same as in the original Java version (MINE.jar, http://www.exploredata.net/). The occurring differences can be ground for small numerical discrepancies as well as for difference in performance (DOI: 10.1093/bioinformatics/bts707).

1. In SOM, Algorithm 5, the characteristic matrix M is computed in the loop starting at line 7 for $xy \leq B$. This is in contrast with the definition of the MINE measures (see SOM, Sec. 2) where the corresponding bound is $xy < B$ for all the four statistics. We adopted the same bound as in the pseudocode, *i.e.* $xy \leq B$.

2. The MINE statistic MCN is defined as follows in SOM, Sec. 2:

$$\text{MCN}(D, \epsilon) = \min_{xy<B} \{\log(xy) \colon M(D)_{x,y} \geq (1-\epsilon)\text{MIC}(D)\}$$

   As for MINE.jar (inferred from Table S1), we set $\epsilon = 0$ and $\log$ to be in base 2. Finally, as specified in Point 1 above, we use the bound $xy \leq B$ as in the SOM pseudocode rather than the $xy < B$ as in the definition. This led to implement the formula:

$$\text{MCN}(D, 0) = \min_{xy\leq B} \{\log_2(xy) \colon M(D)_{x,y} = \text{MIC}(D)\},$$

   being $\text{MIC}(D)$ the maximum value of the matrix $M(D)$.

3. In EquipartitionYAxis() (SOM, Algorithm 3, lines 4 and 10), two ratios are assigned to the variable desiredRowSize, namely $\frac{n}{y}$ and $\frac{(n-i+1)}{(y-\text{currRow}+1)}$. We choose to consider the ratios as real numbers; a possible alternative is to cast desiredRowSize to an integer. The two alternatives can give rise to different Q maps, and thus to slightly different numerical values of the MINE statistics.

4. In some cases, the function EquipartitionYAxis() can return a map Q whose number of clumps $\hat{y}$ is smaller than y, *e.g.* when in D there are enough points whose second coordinates coincide. This can lead to underestimate the normalized mutual information matrix $M_{x,y}$ (SOM, Algorithm 5, line 9), where $M_{x,y}$ is obtained by dividing the mutual information $I_{x,y}$ for $\min\{\log x, \log y\}$. To prevent this issue, we normalize instead by the factor $\min\{\log x, \log \hat{y}\}$.

5. The function GetClumpsPartition(D, Q) is discussed (SOM page 12), but its pseudocode is not explicitely available. Our implementation is defined here in *GetClumpsPartition() algorithm*. The function returns the map P defining the clumps for the set D, with the constraint of keeping in the same clump points with the same x-value.

6. We also explicitly provide the pseudocode for the GetSuperclumpsPartition() function (SOM page 13) in *GetSuperclumpsPartition() algorithm*. This function limits the number of clumps when their number *k* is larger than a given bound $\hat{k}$. The function calls the GetClumpsPartition() and, for math:*k*>*hat{k}* it builds an auxiliary set $D_{\tilde{P}}$ as an input for the EquipartitionYAxis function discussed above (Points 3-4).

Algorithm 1 GetClumpsPartition(D, Q)

Require: $D = \{(a_i, b_i), i = 1, \ldots, n\}$ is a set of n ordered pairs sorted in increasing order by their first component a_i
Require: Q is the map of row assignments returned by EquipartitionYAxis
Ensure: Returns a map $P : D \to \{1, \ldots, k\}$ providing the column assignment of the point (a, b)

```
1:  Q̃ ← Q
2:  i ← 1
3:  c ← −1
4:  repeat
5:      s ← 0
6:      flag ← false
7:      for j = i + 1 to n do
8:          if a_i = a_j then
9:              s ← s + 1
10:             if Q̃((a_i, b_i)) ≠ Q̃((a_j, b_j)) then
11:                 flag ← true
12:     if s ≠ 0 and flag then
13:         for j = 0 to s do
14:             Q̃((a_{i+j}, b_{i+j})) ← c
15:             c ← c − 1
16:     i ← i + s + 1
17: until i > n
18: i ← 1
19: P((a_1, b_1)) ← i
20: for j = 2 to n do
21:     if Q̃((a_j, b_j)) ≠ Q̃((a_{j−1}, b_{j−1})) then
22:         i ← i + 1
23:     P((a_j, b_j)) ← i
24: return P
```

Fig. 3.1: GetClumpsPartition() algorithm

Algorithm 2 GetSuperclumpsPartition($D, Q, \hat{k}$)

Require: $D = \{(a_i, b_i), i = 1, \ldots, n\}$ is a set of n ordered pairs sorted in increasing order by their first component a_i
Require: Q is the map of row assignments returned by EquipartitionYAxis
Require: $\hat{k}$ is the maximum number of clumps
Ensure: Returns a map $P : D \to \{1, \ldots, k\}$ providing the column assignment of the point (a, b)
1: $\tilde{P} \leftarrow$ GetClumpsPartition(D, Q)
2: $k \leftarrow$ number of clumps of $\tilde{P}$
3: **if** $k > \hat{k}$ **then**
4: $\quad D_{\tilde{P}} \leftarrow \{(0, \tilde{P}((a_i, b_i))) : (a_i, b_i) \in D\}$
5: $\quad \hat{P} \leftarrow$ EquipartitionYAxis($D_{\tilde{P}}, \hat{k}$)
6: $\quad P((a_i, b_i)) \leftarrow \hat{P}((0, \tilde{P}((a_i, b_i))))$ for every (a_i, b_i)
7: $\quad$ **return** P
8: **else**
9: $\quad$ **return** $\tilde{P}$

Fig. 3.2: GetSuperclumpsPartition() algorithm

7. We observed that the GetSuperclumpsPartition() implemented in MINE.jar may fail to respect the $\hat{k}$ constraints on the maximum number of clumps and a map P with $\hat{k} + 1$ superclumps is actually returned. As an example, the MINE.jar applied in debug mode (d=4 option) with the same parameters ($\alpha = 0.551$, $c = 10$) used in the original work to the pair of variables (OTU4435,OTU4496) of the Microbioma dataset, returns $cx + 1$ clumps, instead of stopping at the bound $\hat{k} = cx$ for $x = 12, 7, 6, 5, 4 \ldots$.

8. The possibly different implementations of the GetSuperclumpsPartition() function described in Points 6-7 can lead to minor numerical differences in the MIC statistics. To confirm this effect, we verified that by reducing the number of calls to the GetSuperclumpsPartition() algorithm, we can also decrease the difference between MIC computed by minepy and by MINE.jar, and they asymptotically converge to the same value.

9. In our implementation, we use double-precision floating-point numbers (`double` in C) in the computation of entropy and mutual information values. The internal implementation of the same quantities in MINE.jar is unknown.

10. In order to speed up the computation of the MINE statistics, we introduced two improvements (with respect to the pseudo-code), in OptimizeXAxis(), defined in Algorithm 2 in SOM):

    - Given a (P, Q) grid, we precalculate the matrix of number of samples in each cell of the grid, to speed up the computation of entropy values $H(Q)$, $H(\langle c_0, c_s, c_t \rangle)$, $H(\langle c_0, c_s, c_t \rangle, Q)$ and $H(\langle c_s, c_t \rangle, Q)$.
    - We precalculate the entropy matrix $H(\langle c_s, c_t \rangle, Q), \forall s, t$ to speed up the computation of $F(s, t, l)$ (see Algorithm 2, lines 10–17 in SOM).

    These improvements do not affect the final results of mutual information matrix and of MINE statistics.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

C

G

L

M

P

T