PCM

August 16, 2015

Title PCM: A pairwise correlation mining package for biological network inference.
Version 1.0.0
Author Zengyou He, Feiyang Gu and Xiaoqing Liu, Qiong Duan, Bo Tian, Can Zhao, Ben Teng, Jun Wu
Language C++ (include Eigen library)
Description Procedures to pairwise correlation mining using various types of correlation measures. Source codes are available at https://github.com/FeiyangGu/PCM.
Maintainer Zengyou He <zyhe@dlut.edu.cn> Feiyang Gu <gufeiyang1000@gmail.com>

The format of input Columns and rows in the input table correspond to variables (e.g. genes, proteins) and samples (e.g. gene expression profiles), respectively. Each pair of adjacent data in the same row is separated by a space. For binary data, the corresponding values should be 0 or 1. However, there are no additional constraints for continuous data.

The format of output Every row lists a pair of variables whose coefficients satisfy the user-specified constraints and the corresponding value of coefficient. But for LOPCC, there is no specific value of coefficient.

Sample data We provide two groups of sample data. One is a set of network inference data, DREAM3, which hto be 100 rows and 100 columns. DREAM3 is a continuous data set and is named as data_DREAM3.txt in the data file. The other is also a PPI network inference data set, Gavin et al. We transform this data set to a binary table and name it as data_Gavin.txt in the data file. It contains 2166 rows and 2761 columns.

Functions

void SupportB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

SupportB is the implementation of *Support* for binary data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, SupportB will cluster data and then calculate the value. "r" is a parameter that CLOPE needs. SupportB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of SupportB is alternatively, as follows:

```
PCM   SupportB   input.txt   output.txt   0.2
PCM   SupportB   input.txt   output.txt   0.2   1
PCM   SupportB   input.txt   output.txt   0.2   1   1.5
```

void JaccardB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

JaccardB is the implementation of *Jaccard* for binary data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, JaccardB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. JaccardB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of JaccardB is alternatively, as follows:

```
PCM   JaccardB   input.txt   output.txt   0.2
PCM   JaccardB       input.txt   output.txt   0.2   1
PCM   JaccardB       input.txt   output.txt   0.2   1   1.5
```

void InterestB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

InterestB is the implementation of *Interest* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, InterestB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. InterestB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of InterestB is alternatively, as follows:

```
PCM   InterestB       input.txt   output.txt   0.2
PCM   InterestB       input.txt   output.txt   0.2   1
PCM   InterestB       input.txt   output.txt   0.2   1   1.5
```

void Piatetsky_ShapirosB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). Piatetsky_ShapirosB is the implementation of *Piatetsky-Shapiro's* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, Piatetsky_ShapirosB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. Piatetsky_ShapirosB can find all pairs whose Corresponding correlation coefficients are no less than the given threshold. The usage of Piatetsky_ShapirosB is alternatively, as follows:

```
PCM   Piatetsky_ShapirosB       input.txt   output.txt   0.2
PCM   Piatetsky_ShapirosB       input.txt   output.txt   0.2   1
PCM   Piatetsky_ShapirosB       input.txt   output.txt   0.2   1   1.5
```

void CosineB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

CosineB is the implementation of *Cosine* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, CosineB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. CosineB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of CosineB is alternatively, as follows:

```
PCM   CosineB     input.txt     output.txt   0.2
PCM   CosineB     input.txt     output.txt   0.2   1
PCM   CosineB     input.txt     output.txt   0.2   1   1.5
```

void ConfidenceB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

ConfidenceB is the implementation of *Confidence* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, ConfidenceB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. ConfidenceB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of ConfidenceB is alternatively, as follows:

```
PCM   ConfidenceB     input.txt     output.txt   0.2
PCM   ConfidenceB     input.txt     output.txt   0.2   1
PCM   ConfidenceB     input.txt     output.txt   0.2   1   1.5
```

void YulesQB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

YulesQB is the implementation of *Ylues's Q* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, YulesQB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. YulesQB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of YulesQB is alternatively, as follows:

```
PCM   YulesQB    input.txt     output.txt   0.2
PCM   YulesQB    input.txt     output.txt   0.2   1
PCM   YulesQB    input.txt     output.txt   0.2   1   1.5
```

void YulesYB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

YulesYB is the implementation of *Ylues's Y* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, YulesYB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. YulesYB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of YulesYB is alternatively, as follows:

```
PCM   YulesYB    input.txt     output.txt   0.2
PCM   YulesYB    input.txt     output.txt   0.2   1
PCM   YulesYB    input.txt     output.txt   0.2   1   1.5
```

void KappaB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

KappaB is the implementation of *Kappa* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, KappaB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. KappaB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of KappaB is alternatively, as follows:

```
PCM   KappaB     input.txt     output.txt   0.2
PCM   KappaB     input.txt     output.txt   0.2   1
PCM   KappaB     input.txt     output.txt   0.2   1   1.5
```

void J_MeasureB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

J_MeasureB is the implementation of *J-Measure* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, J_MeasureB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. J_MeasureB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of J_MeasureB is alternatively, as follows:

```
PCM   J_MeasureB      input.txt     output.txt   0.2
PCM   J_MeasureB      input.txt     output.txt   0.2   1
PCM   J_MeasureB      input.txt     output.txt   0.2   1   1.5
```

void OddsRatioB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

OddsRatioB is the implementation of *Odds Ratio* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, OddsRatioB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. OddsRatioB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of OddsRatioB is alternatively, as follows:

```
PCM   OddsRatioB      input.txt     output.txt   0.2
PCM   OddsRatioB      input.txt     output.txt   0.2   1
PCM   OddsRatioB      input.txt     output.txt   0.2   1   1.5
```

void GiniIndexB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

GiniIndexB is the implementation of *Gini Index* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, GiniIndexB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. GiniIndexB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of GiniIndexB is alternatively, as follows:

```
PCM   GiniIndexB      input.txt     output.txt   0.2
PCM   GiniIndexB      input.txt     output.txt   0.2   1
PCM   GiniIndexB      input.txt     output.txt   0.2   1   1.5
```

void LaplaceB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

LaplaceB is the implementation of *Laplace* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, LaplaceB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. LaplaceB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of LaplaceB is alternatively, as follows:

```
PCM   LaplaceB    input.txt     output.txt   0.2
PCM   LaplaceB    input.txt     output.txt   0.2   1
PCM   LaplaceB    input.txt     output.txt   0.2   1   1.5
```

void ConvictionB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0).

ConvictionB is the implementation of *Conviction* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, ConvictionB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. ConvictionB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of ConvictionB is alternatively, as follows:

```
PCM    ConvictionB        input.txt       output.txt    0.2
PCM    ConvictionB        input.txt       output.txt    0.2    1
PCM    ConvictionB        input.txt       output.txt    0.2    1    1.5
```

void CertaintyFactorB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). CertaintyFactorB is the implementation of *Certainty Factor* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, CertaintyFactorB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. CertaintyFactorB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of CertaintyFactorB is alternatively, as follows:

```
PCM    CertaintyFactorB        input.txt       output.txt    0.2
PCM    CertaintyFactorB        input.txt       output.txt    0.2    1
PCM    CertaintyFactorB        input.txt       output.txt    0.2    1    1.5
```

void AddedValueB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). AddedValueB is the implementation of *Added Value* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, AddedValueB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. AddedValueB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of AddedValueB is alternatively, as follows:

```
PCM    AddedValueB      input.txt       output.txt    0.2
PCM    AddedValueB      input.txt       output.txt    0.2    1
PCM    AddedValueB      input.txt       output.txt    0.2    1    1.5
```

void CollectiveStrengthB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). CollectiveStrengthB is the implementation of *Collective Strength* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, CollectiveStrengthB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. CollectiveStrengthB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of CollectiveStrengthB is alternatively, as follows:

```
PCM    CollectiveStrengthB      input.txt       output.txt    0.2
PCM    CollectiveStrengthB      input.txt       output.txt    0.2    1
PCM    CollectiveStrengthB      input.txt       output.txt    0.2    1    1.5
```

void KlosgenB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). KlosgenB is the implementation of *Klosgen* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, KlosgenB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. KlosgenB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of KlosgenB is alternatively, as follows:

```
PCM    KlosgenB        input.txt    output.txt   0.2
PCM    KlosgenB        input.txt    output.txt   0.2   1
PCM    KlosgenB        input.txt    output.txt   0.2   1   1.5
```

void Phi_CoefficientB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). Phi_CoefficientB is the implementation of *∅- coefficient* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, Phi_CoefficientB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. Phi_CoefficientB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of Phi_CoefficientB is alternatively, as follows:

```
PCM    Phi_CoefficientB        input.txt    output.txt   0.2
PCM    Phi_CoefficientB        input.txt    output.txt   0.2   1
PCM    Phi_CoefficientB        input.txt    output.txt   0.2   1   1.5
```

void ProbabilityRatioB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). ProbabilityRatioB is the implementation of *Probability Ratio* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, ProbabilityRatioB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. ProbabilityRatioB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of ProbabilityRatioB is alternatively, as follows:

```
PCM    ProbabilityRatioB       input.txt    output.txt   0.2
PCM    ProbabilityRatioB       input.txt    output.txt   0.2   1
PCM    ProbabilityRatioB       input.txt    output.txt   0.2   1   1.5
```

void LikelihoodRatioB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). LikelihoodRatioB is the implementation of *Likelihood Ratio* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, LikelihoodRatioB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. LikelihoodRatioB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of LikelihoodRatioB is alternatively, as follows:

```
PCM    LikelihoodRatioB        input.txt    output.txt   0.2
PCM    LikelihoodRatioB        input.txt    output.txt   0.2   1
PCM    LikelihoodRatioB        input.txt    output.txt   0.2   1   1.5
```

void BCPNNB (char * inputfile, char * outputfile, double threshold, double cc, int kind=0, double r=2.0). BCPNNB is the implementation of *BCPNN* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. The default value of cc is 0.5/n (n is the number of rows). To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, BCPNNB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. BCPNNB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of BCPNNB is alternatively, as follows:

```
PCM   BCPNNB         input.txt    output.txt   0.2
PCM   BCPNNB         input.txt    output.txt   0.2    0.05
PCM   BCPNNB         input.txt    output.txt   0.2    1    1.5
PCM   BCPNNB         input.txt    output.txt   0.2    0.05   1    1.5
```

void SCWCCB (char * inputfile, char * outputfile, double threshold, double cc, int kind=0, double r=2.0). SCWCCB is the implementation of *SCWCC* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. The default value of cc is 0.5/n (n is the number of rows). To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, SCWCCB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. SCWCCB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of SCWCCB is alternatively, as follows:

```
PCM   SCWCCB         input.txt    output.txt   0.2
PCM   SCWCCB         input.txt    output.txt   0.2    0.05
PCM   SCWCCB         input.txt    output.txt   0.2    1    1.5
PCM   SCWCCB         input.txt    output.txt   0.2    0.05   1    1.5
```

void TwoWaySupportB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). TwoWaySupportB is the implementation of *Two-way Support* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, TwoWaySupportB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. TwoWaySupportB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of TwoWaySupportB is alternatively, as follows:

```
PCM   TwoWaySupportB     input.txt    output.txt   0.2
PCM   TwoWaySupportB     input.txt    output.txt   0.2    1
PCM   TwoWaySupportB     input.txt    output.txt   0.2    1    1.5
```

void SCWSB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). SCWSB is the implementation of *SCWS* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, SCWSB will cluster data and then calculate the value. “r” is a parameter that CLOPE needs. SCWSB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of SCWSB is alternatively, as follows:

```
PCM   SCWSB     input.txt    output.txt   0.2
PCM   SCWSB     input.txt    output.txt   0.2    1
PCM   SCWSB     input.txt    output.txt   0.2    1    1.5
```

void SimplifiedXstatisticB (char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). SimplifiedXstatisticB is the implementation of *Simplified* χ*2-statistic* for binary data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, SimplifiedXstatisticB will cluster data and then calculate the value. "r" is a parameter that CLOPE needs. SimplifiedXstatisticB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of SimplifiedXstatisticB is alternatively, as follows:

```
PCM   SimplifiedXstatisticB     input.txt    output.txt   0.2
PCM   SimplifiedXstatisticB     input.txt    output.txt   0.2   1
PCM   SimplifiedXstatisticB     input.txt    output.txt   0.2   1   1.5
```

void MIB(char * inputfile, char * outputfile, double threshold, int kind=0, double r=2.0). MIB is the implementation of *MI (Mutual Information)* for binary data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the CLOPE algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, MIB will cluster data and then calculate the value. "r" is a parameter that CLOPE needs. MIB can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of MIB is alternatively, as follows:

```
PCM   MIB      input.txt     output.txt   0.2
PCM   MIB      input.txt     output.txt   0.2   1
PCM   MIB      input.txt     output.txt   0.2   1   1.5
```

void CMI2NIB(char * inputfile, char * outputfile, double threshold, int order0 = 1). CMI2NIB is the implementation of *CMI2NI* for binary data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. The default value of "order0" is 1. The usage of CMI2NIB is alternatively, as follows:

```
PCM   CMI2NIB   input.txt   output.txt   0.02
PCM   CMI2NIB   input.txt   output.txt   0.02    1
```

void DiceC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4). DiceC is the implementation of Dice for continuous data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, DiceC will cluster data and then calculate the value. "k" is a parameter that k-means needs. DiceC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of DiceC is alternatively, as follows:

```
PCM   DiceC    input.txt     output.txt   0.2
PCM   DiceC    input.txt     output.txt   0.2   1
PCM   DiceC    input.txt     output.txt   0.2   1   5
```

void OverlapC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4). OverlapC is the implementation of *Overlap* for continuous data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, OverlapC will cluster data and then calculate the value. "k" is a parameter that k-means needs. OverlapC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of OverlapC is alternatively, as follows:

```
PCM    OverlapC         input.txt        output.txt    0.2
PCM    OverlapC         input.txt        output.txt    0.2    1
PCM    OverlapC         input.txt        output.txt    0.2    1    5
```

void CosineC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4). CosineC is the implementation of *Cosine* for continuous data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, CosineC will cluster data and then calculate the value. “k” is a parameter that k-means needs. CosineC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of CosineC is alternatively, as follows:

```
PCM    CosineC    input.txt        output.txt    0.2
PCM    CosineC    input.txt        output.txt    0.2    1
PCM    CosineC    input.txt        output.txt    0.2    1    5
```

void JaccardC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4). JarrardC is the implementation of *Jaccard* for continuous data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, JarrardC will cluster data and then calculate the value. “k” is a parameter that k-means needs. JarrardC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of JarrardC is alternatively, as follows:

```
PCM    JarrardC    input.txt        output.txt    0.2
PCM    JarrardC    input.txt        output.txt    0.2    1
PCM    JarrardC    input.txt        output.txt    0.2    1    5
```

void PearsonC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4). PearsonC is the implementation of *Pearson* for continuous data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, PearsonC will cluster data and then calculate the value. “k” is a parameter that k-means needs. PearsonC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of PearsonC is alternatively, as follows:

```
PCM    PearsonC    input.txt        output.txt    0.2
PCM    PearsonC    input.txt        output.txt    0.2    1
PCM    PearsonC    input.txt        output.txt    0.2    1    5
```

void SpearmanC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4). SpearmanC is the implementation of *Spearman* for continuous data. “inputfile” and “outputfile” are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of “kind” is 0, which means users immediately calculate the correlation without clustering. If users set “kind” to be 1, SpearmanC will cluster data and then calculate the value. “k” is a parameter that k-means needs. SpearmanC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of SpearmanC is alternatively, as follows:

```
PCM    SpearmanC      input.txt      output.txt   0.2
PCM    SpearmanC      input.txt      output.txt   0.2   1
PCM    SpearmanC      input.txt      output.txt   0.2   1   5
```

void DotProductC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4).

DotProductC is the implementation of *Dot Product* for continuous data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, DotProductC will cluster data and then calculate the value. "k" is a parameter that k-means needs. DotProductC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of DiceC is alternatively, as follows:

```
PCM    DotProductC    input.txt      output.txt   0.2
PCM    DotProductC    input.txt      output.txt   0.2   1
PCM    DotProductC    input.txt      output.txt   0.2   1   5
```

void KendallC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4). KendallC is the implementation of *Kendall Rank* for continuous data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, KendallC will cluster data and then calculate the value. "k" is a parameter that k-means needs. KendallC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of KendallC is alternatively, as follows:

```
PCM    KendallC   input.txt      output.txt   0.2
PCM    KendallC   input.txt      output.txt   0.2   1
PCM    KendallC   input.txt      output.txt   0.2   1   5
```

void HoeffdingDC (char * inputfile, char * outputfile, double threshold, int kind=0, int k=4).

HoeffdingDC is the implementation of *Hoeffding's D measure* for continuous data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. To speed up the efficiency, we utilize the k-means algorithm to cluster the data first and then calculate the pairs which are in the same cluster. The default value of "kind" is 0, which means users immediately calculate the correlation without clustering. If users set "kind" to be 1, HoeffdingDC will cluster data and then calculate the value. "k" is a parameter that k-means needs. HoeffdingDC can find all pairs whose corresponding correlation coefficients are no less than the given threshold. The usage of HoeffdingDC is alternatively, as follows:

```
PCM    HoeffdingDC    input.txt      output.txt   0.2
PCM    HoeffdingDC    input.txt      output.txt   0.2   1
PCM    HoeffdingDC    input.txt      output.txt   0.2   1   5
```

void CMI2NIC(char * inputfile, char * outputfile, double threshold, int order0 = 2). CMI2NIC is the implementation of *CMI2NI* for continuous data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. The default value of order0 is 2. The usage of CMI2NIC is alternatively, as follows:

```
PCM    CMI2NIB    input.txt    output.txt    0.02
PCM    CMI2NIB    input.txt    output.txt    0.02     2
```

void LOPCC(char * inputfile, char * outputfile). LOPCC is the implementation of *LOPC* for continuous data. "inputfile" and "outputfile" are the file paths of input data and output data, respectively. The usage of LOPCC is alternatively, as follows:

```
PCM    LOPCC input.txt    output.txt
```

We also provide some other interfaces which can calculate the correlation between two variables for further convenient use.

The interfaces with names prefixed with “BK” are for binary data and the other interfaces which begin with “C”" are for continuous data.

The interfaces are:

```
double BK_Support(vector<int>& var1, vector<int>& var2);
double BK_Jaccard(vector<int>& var1, vector<int>& var2);
double BK_Interest(vector<int>& var1, vector<int>& var2);
double BK_Piatetsky_Shapiros(vector<int>& var1, vector<int>& var2);
double BK_Cosine(vector<int>& var1, vector<int>& var2);
double BK_Confidence(vector<int>& var1, vector<int>& var2);
double BK_YulesQ(vector<int>& var1, vector<int>& var2);
double BK_YulesY(vector<int>& var1, vector<int>& var2);
double BK_Kappa(vector<int>& var1, vector<int>& var2);
double BK_J_Measure(vector<int>& var1, vector<int>& var2);
double BK_OddsRatio(vector<int>& var1, vector<int>& var2);
double BK_GiniIndex(vector<int>& var1, vector<int>& var2);
double BK_Laplace(vector<int>& var1, vector<int>& var2);
double BK_Conviction(vector<int>& var1, vector<int>& var2);
double BK_CertaintyFactor(vector<int>& var1, vector<int>& var2);
double BK_AddedValue(vector<int>& var1, vector<int>& var2);
double BK_CollectiveStrength(vector<int>& var1, vector<int>& var2);
double BK_Klosgen(vector<int>& var1, vector<int>& var2);
double BK_Phi_Coefficient(vector<int>& var1, vector<int>& var2);
double BK_ProbabilityRatio(vector<int>& var1, vector<int>& var2);
double BK_LikelihoodRatio(vector<int>& var1, vector<int>& var2);
double BK_BCPNN(vector<int>& var1, vector<int>& var2, double cc);
double BK_SCWCC(vector<int>& var1, vector<int>& var2, double cc);
double BK_TwoWaySupport(vector<int>& var1, vector<int>& var2);
double BK_SCWS(vector<int>& var1, vector<int>& var2);
double BK_SimplifiedXstatistic(vector<int>& var1, vector<int>& var2);

double CK_Dice(vector<double>& var1, vector<double>& var2);
double CK_Jaccard(vector<double>& var1, vector<double>& var2);
double CK_Overlap(vector<double>& var1, vector<double>& var2);
double CK_Cosin(vector<double>& var1, vector<double>& var2);
double CK_Pearson(vector<double>& var1, vector<double>& var2);
double CK_Spearman(vector<double>& var1, vector<double>& var2);
double CK_DotProduct(vector<double>& var1, vector<double>& var2);
double CK_Kendall(vector<double>& var1, vector<double>& var2);
double CK_Distance(vector<double>& var1, vector<double>& var2);
double CK_HoeffdingD(vector<double>& var1, vector<double>& var2);
```