ドキュメント

条件付きガウススコアの実装と完全混合データからの

ベイジアンネットワークの学習

プログラムの概要

あらまし

ベイジアンネットワークは色々な要素がどういうふうに関係しているかを、視覚的に表すことができる。実際に、医療診断システムなど多くの推論に応用されており、観測データからベイジアンネットワークの構造を自動的に学習する方法が多くの研究者に研究されてきた。私は、pygobnilp という、データからベイジアンネットワークを学習するための Python ライブラリの改善に取り組んだ。このライブラリは、可能なネットワークの構造の尤度をデータから計算し、この尤度が最大になる、つまり与えられたデータをもっともらしく説明しているネットワークを整数計画問題にモデル化することで推定する。しかしこのライブラリは、数値と「はい」や「いいえ」などの離散値が混在した混合データを扱うことができなかった。そこで、混合データからベイジアンネットワークの構造を評価できる手法を pygobnilp に実装した。条件付きガウススコアを計算することで、あらゆる種類のデータから任意の形のベイジアンネットワークを評価することを実現した。

インストール

実行環境の構築には、幾つかのライブラリとソフトウェアを必要とする。

- Python のライブラリ：scipy, pygraphviz, matplotlib, networkx, pandas, numpy, scikit-learn, numba, jupyter（必要に応じて）
- Gurobi MIP solver（研究では Academic License を利用したが、Restricted License でも動く。ただし実行結果が一部異なる）
- Graphviz

最も簡単な方法は Anaconda Python を使った方法である。Anaconda をインストール後、必要なライブラリをインストールし、以下のコマンドを端末に入力することで Python 用の gurobi を使うことができる（詳しくは https://www.gurobi.com/get-anaconda/）。

```
conda config --add channels http://conda.anaconda.org/gurobi
conda install gurobi
```

最後に、ソースコードをワークスペースにダウンロード後、以下のコマンドによりインストールすることができる。

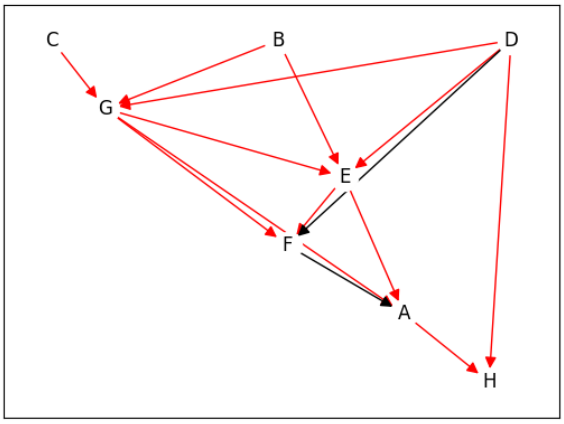
```
python pygobnilp/setup.py develop
```

実行例

実行例は Git 上のノートブックにある（https://github.com/2017Yasu/newPygobnilp/blob/main/testMixedDataLearning.ipynb）。また、スクリプトから以下のように実行することも可能である（オプションの詳細は https://www.cs.york.ac.uk/aig/sw/gobnilp/pygobnilpmanual.pdf）。

```
$ python pygobnilp/rungobnilp.py data/clgaussian.test --score CGaussianBIC --palim 3
Academic license - for non-commercial use only - expires 2021-07-13
Using license file ***/gurobi.lic
Changed value of parameter PreCrush to 1
   Prev: 0  Min: 0  Max: 1  Default: 0
Changed value of parameter CutPasses to 100000
   Prev: -1  Min: -1  Max: 2000000000  Default: -1
Changed value of parameter GomoryPasses to 100000
   Prev: -1  Min: -1  Max: 2000000000  Default: -1
Changed value of parameter MIPFocus to 2
   Prev: 0  Min: 0  Max: 3  Default: 0
Changed value of parameter ZeroHalfCuts to 2
   Prev: -1  Min: -1  Max: 2  Default: -1
Changed value of parameter MIPGap to 0.0
   Prev: 0.0001  Min: 0.0  Max: inf  Default: 0.0001
Changed value of parameter MIPGapAbs to 0.0
   Prev: 1e-10  Min: 0.0  Max: inf  Default: 1e-10
**********
BN has score -26192.129205509955
**********
A<-E,F,G 8443.560026235356
G<-B,C,D -18158.74739297181
E<-B,D,G -6543.9972512624945
F<-D,E,G 7158.322853437495
B<- -5239.826207064481
C<- -6474.285804788533
D<- -9136.136735566384
H<-A,D 3758.9813064708965
**********
```

bnlearn modelstring =
[A|G:E:F][G|C:D:B][E|D:G:B][F|D:G:E][B][C][D][H|D:A]
****
CPDAG:
Vertices: A,G,E,F,B,C,D,H
A->H
G->A
G->E
G->F
E->A
E->F
F-A
B->E
B->G
C->G
D->E
D-F
D->G
D->H



この実行例では、clgaussian.test にある混合データから、BIC によるペナルティ付きの条件付きガウススコアを使って、親ノードの数を 3 個以下に限定したときの学習結果を示している。

ベイジアンネットワークとは

ベイジアンネットワークは、図 1 に示すような有向非巡回グラフ（DAG）$G = (V, E)$ で表され、n 個のランダム変数 $V = \{X_1, X_2, \ldots, X_n\}$ と変数間の依存関係を示す有向辺の集合 E を持つ。変数はそれぞれ離散値か連続値を取る。有向辺 $(X_i, X_j) \in E$ は変数 X_j が直接的に X_i の影響を受けることを表し、X_i を X_j の親ノード、X_j を X_i の子ノードと呼ぶ。図 1 の例では、変数 D は親ノードである変数 E，B の子ノードであり、視覚的に D は E と B の影響を受けていることを示している。

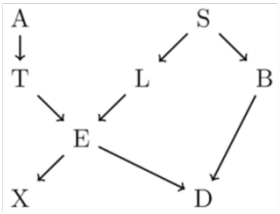


図 1．8 個の変数を持つベイジアンネットワークの有向非巡回グラフ

ベイジアンネットワークはさらに、各変数の条件付き確率分布から成るパラメータセット $\boldsymbol{\theta}$ を持つ。例えば、図 1 の各変数が $true$ または $false$ の 2 値を取る場合、変数 D のパラメータは以下のような条件付き確率表（CPT）により表すことができる。

$$\begin{cases} P(D = t \mid B = f, E = f) = 0.2 \\ P(D = f \mid B = f, E = f) = 0.8 \\ P(D = t \mid B = f, E = t) = 0.6 \\ P(D = f \mid B = f, E = t) = 0.4 \\ P(D = t \mid B = t, E = f) = 0.1 \\ P(D = f \mid B = t, E = f) = 0.9 \\ P(D = t \mid B = t, E = t) = 1.0 \\ P(D = f \mid B = t, E = t) = 0.0 \end{cases}$$

変数が連続値を取る場合のパラメータは、線形回帰モデルにより予測される関数の係数により表すことができる。

混合データからベイジアンネットワークの DAG を学習する方法

整数計画問題にモデル化することで、学習を行う。ここでは、整数計画問題の変数を $I(W \to v)$ として、変数 v の親ノードの集合が W であるときは 1，違う場合は 0 と定義する。このとき、与えられたデータを $Data$ とすると、整数計画問題により最大化する目的関数は以下のように表される。

$$S(G \mid Data) = \sum_{v,W} s(v, W) I(W \to v)$$

この式により求められる値を特にグローバルスコアと呼び、$s(v, W)$ をローカルスコアと

呼ぶ。

グローバルスコアを求めるには、いくつかの方法があるが、ここでは混合データから学習する時に利用できる条件付きガウススコアについて簡単に説明する。i 番目の変数を Y_i 、その親ノードの集合を Pa_i とする。また、Pa_i のうち離散値を取る変数の集合を Pd_i 、連続値を取る変数の集合を Pc_i とする。このとき、子ノード Y_i と親ノードの集合 Pa_i のローカルな条件付きガウススコアは、$Y_i \cup Pa_i$ と Pa_i の対数尤度の差により計算することができる。また、自由度も同様に、$Y_i \cup Pa_i$ と Pa_i の自由度の差により計算できる。

$Y_i \cup Pa_i$ および Pa_i の対数尤度は次のように求める。まず、与えられたデータ $Data$ を離散変数（$Y_i \cup Pd_i$ または Pd_i）の値の組み合わせにより、部分データ $Data_p$ に分割する。このとき、$Data_p$ は次のような行列の形をしている。

$$Data_p = \begin{pmatrix} \boldsymbol{x}_{p,1}^T \\ \boldsymbol{x}_{p,1}^T \\ \vdots \\ \boldsymbol{x}_{p,j}^T \\ \vdots \\ \boldsymbol{x}_{p,n_p}^T \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{j,1} & x_{j,2} & \cdots & x_{j,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n_p,1} & x_{n_p,2} & \cdots & x_{n_p,d} \end{pmatrix}$$

ここで、d は $Data_p$ に含まれている連続変数の数、n_p は $Data_p$ に含まれているデータの数、列ベクトル $\boldsymbol{x}_{p,j}$ は j 番目の観測データを表す。この部分データにおける対数尤度と自由度は次のように計算する。

$$l_p\left(\widehat{\boldsymbol{\theta}}_p \mid Data_p\right) = -\frac{n_p}{2}\left(\log\left|\widehat{\boldsymbol{\Sigma}}_p\right| + d\log 2\pi + d\right) + n_p \log\frac{n_p}{n},$$

$$df_p\left(\widehat{\boldsymbol{\theta}}_p\right) = \frac{d(d+1)}{2} + 1$$

ここで、n は全体のデータ数を表している。また、$\left|\widehat{\boldsymbol{\Sigma}}_p\right|$ は標本共分散行列の行列値を表しており、標本共分散行列は標本平均ベクトル $\bar{\boldsymbol{x}}_p = \frac{1}{n_p}\sum_j^{n_p} \boldsymbol{x}_{p,j}$ を使って以下のように計算できる。

$$\widehat{\boldsymbol{\Sigma}}_p = \frac{1}{n_p}\sum_{j=1}^{n_p}\left(\boldsymbol{x}_{p,j} - \bar{\boldsymbol{x}}_p\right)\left(\boldsymbol{x}_{p,j} - \bar{\boldsymbol{x}}_p\right)^T$$

したがって、$\widehat{\boldsymbol{\Sigma}}_p$ は $d \times d$ の行列となる。各部分データにおける対数尤度と自由度を計算後、それぞれを合計することで $Y_i \cup Pa_i$ および Pa_i の対数尤度と自由度が求まる。最後に、$Y_i \cup Pa_i$ および Pa_i の対数尤度と自由度から、以下のようにして子ノード Y_i と親ノードの集合 Pa_i のローカルな条件付きガウススコアを求めることができる。

$$l_i\left(\widehat{\boldsymbol{\theta}}_p \mid Data_p\right) = l_{Y_i \cup Pa_i}\left(\widehat{\boldsymbol{\theta}}_p \mid Data_p\right) - l_{Pa_i}\left(\widehat{\boldsymbol{\theta}}_p \mid Data_p\right),$$
$$df_i\left(\widehat{\boldsymbol{\theta}}_p\right) = df_{Y_i \cup Pa_i}\left(\widehat{\boldsymbol{\theta}}_p\right) - df_{Pa_i}\left(\widehat{\boldsymbol{\theta}}_p\right)$$

同様にして、与えられた親ノードと子ノードの組み合わせについてローカルスコアをそれぞれ求め、合計することで、あるデータが与えられたときの条件付きガウススコアを求めることができる。
求めた条件付きガウススコアに対して、DAG の複雑性によるペナルティを加えることがある。DAG の複雑性は自由度の合計により計算され、pygobnilp には次の二つのペナルティを含めた条件付きガウススコアが実装されている。

$$BIC: \quad \sum_i l_i(\hat{\boldsymbol{\theta}}_p \mid Data_p) - \frac{k}{2}\log n \sum_i df_i(\hat{\boldsymbol{\theta}}_p)$$
$$AIC: \quad \sum_i l_i(\hat{\boldsymbol{\theta}}_p \mid Data_p) - k \sum_i df_i(\hat{\boldsymbol{\theta}}_p) \qquad (k \text{ is a const.})$$

クラスの説明

ソースコードは Python で記述されている。ここでは特に、新たに作成したクラスについて説明する。それ以外のクラスや関数については、正式なドキュメントを参照（https://pygobnilp.readthedocs.io/en/latest/）。

MixedData クラス

class pygobnilp.scoring.MixedData(data_source, varnames=None, arities=None)

継承：pygobnilp.scoring.Data
説明：混合データを保持するためのクラス。
メソッド：

__init__(data_source, varnames=None, arities=None)

コンストラクタ。MixedData オブジェクトを初期化する。
引数：

- data_source (str/array_like/Pandas.DataFrame)：データファイルの名前、二つの numpy.ndarray を持つタプル、または Pandas のデータフレーム。ファイル名を指定した場合、データファイルは以下を満たしていることを前提とする。
  1. 全てのデータはスペースにより区切られる
  2. コメントは“#”により始まる
  3. 先頭行は変数名を表す
  4. 第二行は各変数のアリティ（離散変数のとりうる値の数，連続変数の場合は“-”とする）を表す
  5. それ以外の部分は実データを表す
- varnames (iter)：各データ列に対応する変数名のリスト。data_source がファイル名または Pandas.DataFrame の場合、これは無視される。指定されなかった場合、離散変数は D1, D2, ...，連続変数は C1, C2, ... と自動的に対応される。

- arities (iter)：各データ列に対応する変数のアリティのリスト。data_source がファイル名または Pandas.DataFrame の場合、これは無視される。指定されなかった場合、出現した値の種類数により決定される。

arities()

各離散変数のアリティを返す。
返り値：numpy.ndarray

arity(v)

与えられた変数のアリティを返す。連続変数が与えられた場合、ValueError を返す。
引数：
- v (str)：変数名。

返り値：int

data()

全てのデータを返す。
返り値：pandas.DataFrame

data_length()

データレコードの数を返す。
返り値：int

rawdata()

変数名を除く離散データ、連続データを両方返す。
返り値：numpy.ndarray, numpy.ndarray

variables()

変数名のリストを返す。
返り値：list

AbsCGaussianLLScore クラス

class pygobnilp.scoring.AbsCGaussianLLScore(data_source, varnames=None, arities=None)

継承：pygobnilp.scoring.MixedData
説明：混合データからベイジアンネットワークの条件付きガウススコアを計算するための

抽象クラス。

メソッド：

c_gaussianll(variables)

与えられた変数リストの対数尤度と自由度を計算する。

引数：

- variables (iter)：変数の集合

返り値：tuple(float, int)

con_gaussianll(indices)

与えられたインデックスに対応する連続変数の、対数尤度と自由度を計算する。注目している変数が全て連続変数の場合のみ利用される。

引数：

- indices (iter)：連続変数のインデックスの集合

返り値：tuple(float, int)

dis_gaussian(indices)

与えられたインデックスに対応する離散変数の、対数尤度と自由度を計算する。注目している変数が全て離散変数の場合のみ利用される。

引数：

- indices (iter)：離散変数のインデックスの集合

返り値：tuple(float, int)

ll_score(child, parents)

child で与えられた変数が、parents で与えられた複数の変数を親ノードとして持つときの、ローカルな条件付きガウススコアとその自由度を計算する。

引数：

- child (str)：子ノードの変数名
- parents (iter)：親ノードの変数名のリスト

返り値：tuple(float, int)

mix_gaussianll(cindices, rindices)

与えられたインデックスに対応する離散変数の対数尤度と自由度を、rindices により与えられた部分データのみを利用して計算する。注目している変数に離散変数と連続変数が含まれている場合のみ利用される。

引数：

- cindices (iter)：連続変数のインデックスの集合
- rindices (iter)：データの一部を抽出するためのインデックスの集合

返り値：tuple(float, int)

CGaussianLL クラス

class pygobnilp.scoring.CGaussianLL(data)

継承：pygobnilp.scoring.AbsCGaussianLLScore

説明：条件付きガウススコアを計算するためのクラス。DAG の複雑性によるペナルティは計算しない。

メソッド：

__init__(data)

コンストラクタ。CGaussianLL オブジェクトを初期化する。

引数：

- data (MixedData)：混合データのオブジェクト

score(child, parents)

child で与えられた変数が、parents で与えられた複数の変数を親ノードとして持つときの、ローカルな条件付きガウススコアを計算する。また、child で与えられた変数がとりうる条件付きガウススコアの上限値を返す。

引数：

- child (str)：子ノードの変数名
- parents (iter)：親ノードの変数名のリスト

返り値：tuple(float, float)

score_dag(dag, by_node=False)

与えられた DAG における、各変数のローカルな条件付きガウススコアを計算し、それを合計することでグローバルなスコアを計算する。

引数：

- dag (str)：DAG を表した文字列モデル（例：[A][B][C][D|F:E:G][F][E|A:B][G|A:F:E][H|A:D:B]）。
- by_node (bool)：False（デフォルト）の場合、与えられた DAG のグローバルなスコアを計算返す。True の場合、各変数のローカルなスコアを辞書型で返す。

返り値：float または dict

CGaussianBIC クラス

class pygobnilp.scoring.CGaussianBIC(data, k=1)

継承：pygobnilp.scoring.AbsCGaussianLLScore

説明：条件付きガウススコアから、DAG の複雑性をペナルティとして $\frac{k}{2}\log N \times df$ をマイナスすることで BIC スコアを計算する。

メソッド：

__init__(data, k=1)

コンストラクタ。CGaussianBIC オブジェクトを初期化する。

引数：

- data (MixedData)：混合データのオブジェクト
- k (float)：ペナルティの影響の強さ

score(child, parents)

child で与えられた変数が、parents で与えられた複数の変数を親ノードとして持つときの、ローカルな BIC スコアを計算する。また、child で与えられた変数がとりうる BIC スコアの上限値を返す。

引数：

- child (str)：子ノードの変数名
- parents (iter)：親ノードの変数名のリスト

返り値：tuple(float, float)

CGaussianAIC クラス

class pygobnilp.scoring.CGaussianAIC(data, k=1)

継承：pygobnilp.scoring.AbsCGaussianLLScore

説明：条件付きガウススコアから、DAG の複雑性をペナルティとして $k \times df$ をマイナスすることで AIC スコアを計算する。

メソッド：

__init__(data, k=1)

コンストラクタ。CGaussianAIC オブジェクトを初期化する。

引数：

- data (MixedData)：混合データのオブジェクト
- k (float)：ペナルティの影響の強さ

score(child, parents)

child で与えられた変数が、parents で与えられた複数の変数を親ノードとして持つと

きの、ローカルな AIC スコアを計算する。また、child で与えられた変数がとりうる AIC スコアの上限値を返す。

引数：

- child (str)：子ノードの変数名
- parents (iter)：親ノードの変数名のリスト

返り値：tuple(float, float)

実験

ここでは、以下の 4 点について実験したことを簡単に説明する。

1. ローカルな条件付きガウススコアの確認
2. データサイズによる計算速度の変化
3. 変数の数による計算速度の変化
4. ベイジアンネットワークの学習精度

なお、この実験では R パッケージの bnlearn で計算、学習した結果と比較した。Bnlearn は R によりプログラムされた、ベイジアンネットワークのヒューリスティックな学習パッケージである（詳細は https://www.bnlearn.com/）。これは混合データからも学習を行うことができるが、離散変数が連続変数を親ノードとして持つことを禁止しており、任意のベイジアンネットワークを学習することができないという欠点がある。

この実験では、macOS Catalina を利用した（他に MacOS Big Sur と Windows 10 においての動作を確認済み）。実験で利用したコンピュータは、1.2GHz のクアッドコア Intel Core i7 のプロセッサで構成されており、メモリサイズは 16GB であった。Anaconda 1.7.2 のプラットフォームにおいて実験を行った。Pygobnilp は Python 3.7.7 により実行され、bnlearn は R 3.6.1 により実行された。

ローカルな条件付きガウススコアの確認

この実験では、95 種類の親ノードと子ノードの組み合わせにおいて、ローカルな条件付きガウススコアを計算した。データは clgaussian.test を使い、これには 4 つの離散変数 (A, B, C, F) と 4 つの連続変数 (D, E, G, H) が含まれている。表 1 に計算結果の一部を示す。

スコアを比較すると、差は最も大きくて 0.127、相対誤差は 0.007%であることがわかった。比較的小さな誤差であったので、正しく実装できたと結論づけられる。この誤差については、pygobnilp は標本分散、bnlearn では不偏分散を利用して計算しているためだと考えられる。

表 1. データセット clgaussian.test から計算される 12 組のローカルな条件付きガウススコア（$\{X,Y\} \to Z$ は子ノード Z が親ノード $\{X,Y\}$ を持っていることを示す。）

Family	pygobnilp	bnlearn	difference
$\emptyset \to A$	-1567.524	-1567.524	0.000
$\{A,B\} \to D$	-1757.816	-1757.822	0.006
$\{A,F\} \to D$	-1760.120	-1760.122	0.002
$\{A,B,C\} \to D$	-1738.733	-1738.860	0.127
$\{B,F,G\} \to D$	1510.355	1510.346	0.009
$\{A,B,F\} \to E$	-7933.648	-7933.676	0.028
$\{B,D,F\} \to E$	-6531.378	-6531.387	0.009
$\{A,B,F\} \to G$	-11444.497	-11444.525	0.028
$\{A\} \to H$	3447.735	3447.735	0.001
$\{A,C,D\} \to H$	3780.858	3780.809	0.049
$\{A,B,D\} \to H$	3784.368	3784.343	0.025
$\{A,E,G\} \to H$	3463.099	3463.094	0.005

データサイズによる計算速度の変化

データサイズを 5,000 から 2,560,000 まで変化させたとき、1,000 個のベイジアンネットワークの条件付きガウススコアを計算するのにかかった時間は以下のように変化した。

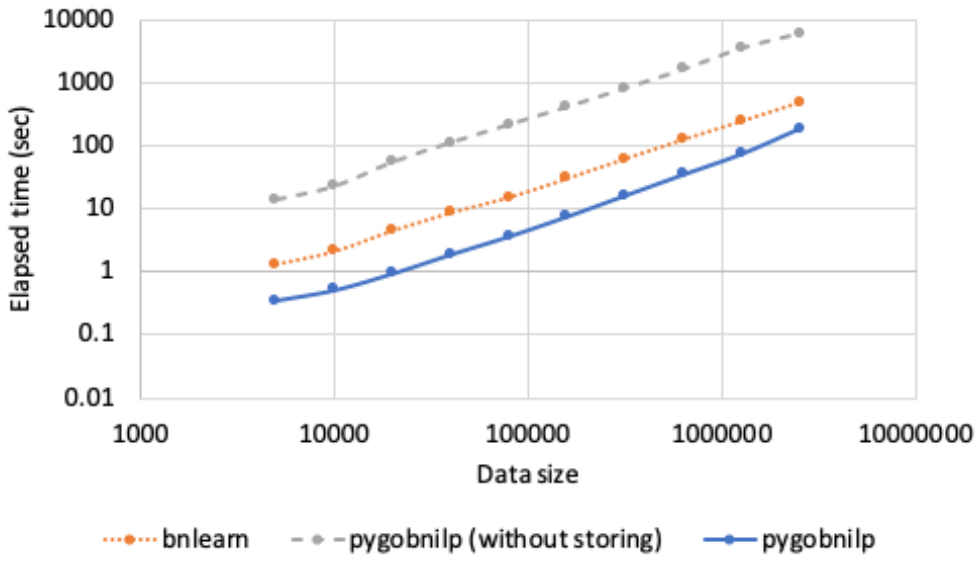


図 2. 条件付きガウススコアの計算時間の比較（“without storing” は計算中に対数尤度をメモリに保存しなかったことを示す）

単純な計算手法では、bnlearn の方が pygobnilp よりも早かった。これは、bnlearn がコンパイルされた C のプログラムを利用していたことが原因として考えられる。また、計算中に対数尤度を覚えておくことで、計算が劇的に早くなることがわかった。

変数の数による計算速度の変化

次に、変数の数を 8 から 20 まで変化させて、全ての子ノードと親ノードの組み合わせのローカルスコアを計算したときの時間を計測した。この実験について、bnlearn は個別にローカルスコアを計算することができないので、pygobnilp での実験結果についてのみ言及する。

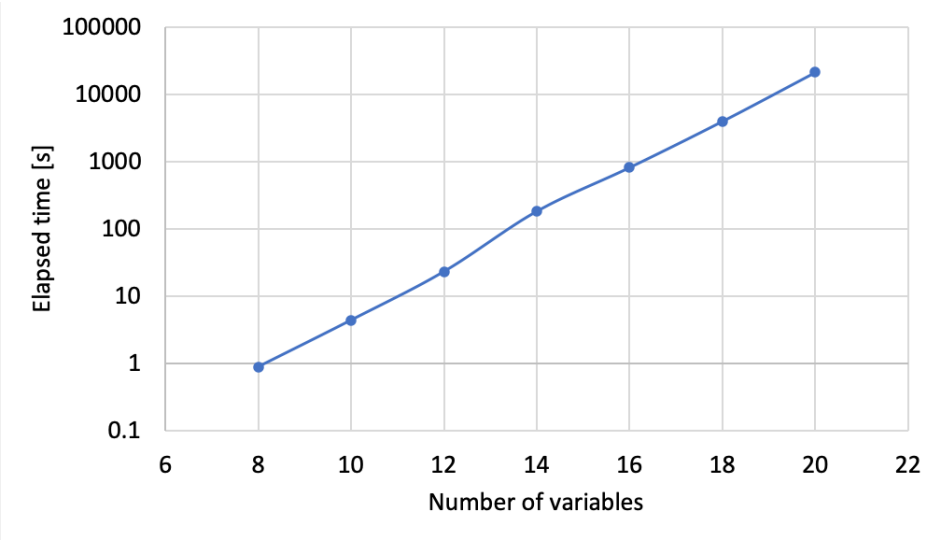


図 3. 変数の数を変化させたときの条件付きガウススコアの計算にかかる時間の変化

この結果から、変数の数が増えるにつれて指数関数的に長い時間を必要とすることがわかった。理論的にも、変数の数を $|V|$ とすると $O(2^{|V|})$ の計算を必要とするので、この結果は妥当であると言える。

ベイジアンネットワークの学習精度

最後に、学習精度についての実験を行なった。この実験では、表 2 と表 3 に示したベイジアンネットワークからランダムに生成したデータを使って学習行い、どれだけ近い DAG を学習することができるのかを評価した。データの生成には、“tetrad”というソフトウェアを利用し（http://www.phil.cmu.edu/tetrad/）、tetraddata_20_1.txt と tetraddata_20_2.txt に保存してある。学習精度の指標には SHD (Structural Hamming Distance) を利用した。これは、学習した DAG を正解の DAG と一致させるために、追加、削除、変更する必要がある有向辺の本数である。

表 2. データ生成に利用したベイジアンネットワークの各変数名とそれぞれの親ノード（“C”は連続変数、“D”は離散変数を示す。これは連続変数から離散変数への有向辺が存在する）

Variable	Type	Parents	Variable	Type	Parents
X1	C	∅	X11	D	X2, X3
X2	D	∅	X12	C	X7, X8
X3	C	∅	X13	C	X6, X11, X12
X4	D	X3	X14	C	X7, X11, X13
X5	C	X1, X2	X15	C	X1, X12, X14
X6	D	X1, X4	X16	D	X5, X6, X10
X7	D	∅	X17	D	X2, X9, X14
X8	C	X5	X18	D	X3, X7, X16
X9	D	X5, X6	X19	C	X18
X10	D	X4, X8, X9	X20	C	X4, X8, X12

表 3. データ生成に利用したベイジアンネットワークの各変数名とそれぞれの親ノード（“C”は連続変数、“D”は離散変数を示す。これは連続変数から離散変数への有向辺が存在しない）

Variable	Type	Parents	Variable	Type	Parents
X1	D	∅	X11	C	X2, X3
X2	D	X1, X5, X12	X12	D	X5, X7, X19
X3	C	X5, X9, X18	X13	D	X8, X19
X4	C	X10, X17	X14	C	X1, X4
X5	D	X1	X15	C	X7, X9
X6	C	X5, X9, X13	X16	C	X1, X4
X7	D	X5, X9, X10	X17	C	X1, X9
X8	D	X5, X7, X10	X18	C	X5, X16
X9	D	X1, X5	X19	D	X8, X10
X10	D	X1, X5, X9	X20	C	X10, X17

以下の表 4 と表 5 に、学習した DAG と正解の DAG との比較結果を示す。
これらの結果から、pygobnilp の方が bnlearn よりもより精度の高い学習を実現することができたと結論づけられる。DAG に連続変数から離散変数への有向辺が存在しない場合でも pygobnilp の方高い精度だった理由として、bnlearn ではヒューリスティック探索を利用していることが挙げられる。

表 4. 学習された DAG と正解の DAG との間の SHD（正解の DAG には連続変数から離散変数への有向辺が存在する）

	Pygobnilp	bnlearn
CG	23	31
BIC	6	18
AIC	16	24

表 5. 学習された DAG と正解の DAG との間の SHD（正解の DAG には連続変数から離散変数への有向辺が存在しない）

	Pygobnilp	bnlearn
CG	21	34
BIC	21	29
AIC	29	35

自己評価

この研究を通して、混合データからのベイジアンネットワークの学習を可能にし、ライブラリ pygobnilp の改善に貢献することができた。今後の研究として、このライブラリが実際に幅広いモデルに対して有効な学習を行うことができることを、より多くのベイジアンネットワークを使って確かめる必要がある。また、多くの変数が存在するとき、指数関数的に長い時間がかかっていたので、より良い最適化手法が必要だと考えられる。
個人的には、多くの参考文献を自分で探し、学習することで、研究目的を実現することができたと考えている。実際のプログラムも、自分でテストした範囲ではバグはないと考えられる。担当教員以外からの評価は 100 点中 78 点で、かなり高い成績を得ることができ、満足している。しかし、ピアレビューなどをしていないので、例外処理などの更なる改善が必要であると考えられる。

参考文献

[1] W. Wiegerinck, et al., “Bayesian networks for expert systems: theory and practical applications,” in *Interactive collaborative information systems*, Springer, 2010, pp. 547-578.
[2] M. Barlett and J. Cussens, “Advances in Bayesian Network Learning Using Integer Programming,” *ArXiv.org*, March 23, 2015.
[3] M. Scutari, *bnlearn-Bayesian network structure learning*, bnlearn.com. [Online]. Available: https://www.bnlearn.com/. [Accessed: 21 June 2020].
[4] C. Glymour, et al. (2015), *Tetrad*, Carnegie Mellon University. [Online]. Available: http://www.phil.cmu.edu/tetrad/. [Accessed: 21 June 2020].

[5] University of York (2017, Jan.), *GOBNILP*, cs.york.ac.uk. [Online]. Available: https://www.cs.york.ac.uk/aig/sw/gobnilp/. [Accessed: 21 June 2020].
[6] J. Cussens, "Integer Programming for Bayesian Network Structure Learning," *Quality Technology & Quantitative Management*, vol. 11, no. 1, pp. 99–110, 2014.
[7] A. M. Carvalho, "Scoring functions for learning Bayesian networks," INESC-ID Tec, Rep. 54/2009, April 2009.
[8] S. S. Qian and R. J. Miltner, "A continuous variable Bayesian networks model for water quality modeling: A case study of setting nitrogen criterion for small rivers and streams in Ohio, USA," *Environmental Modelling & Software*, vol. 69, pp. 14-22, 2015.
[9] Z. Liu, et al., "Empirical Evaluation of Scoring Functions for Bayesian Network Model Selection," *BMC Bioinformatics*, vol. 13, no. Suppl 15, 2012.
[10] Y. C. Chen, et al., "Learning discrete Bayesian networks from continuous data," *Artificial Intelligence Research*, vol. 59, pp. 103-132, 2017.
[11] B. Andrews, et al., "Scoring Bayesian networks of mixed variables," *International Journal of Data Science and Analytics*, vol. 6, no. 1, pp. 3–18, 2018.
[12] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Royal Statistical Society: Series B (Methodological)*, vol. 50, no. 2, pp. 157-294, January 1988.
[13] H. Pishro-Nik (2014), *Conditional independence.* Kappa Research LLC [Online]. Available: https://www.probabilitycourse.com/chapter1/1_4_4_conditional_independence.php. [Accessed: Aug. 17, 2020].
[14] R. Trotta, "Bayes in the sky: Bayesian inference and model selection in cosmology," *Contemporary Physics*, vol. 49, no. 2, pp. 71-104, 2008.
[15] T. Jaakkola, et al., "Learning Bayesian network structure using LP relaxations." in *Proceedings of the Thirteenth Informational Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, Italy, 2010. pp. 358-365.
[16] Gurobi. *Gurobi - The fastest solver*, gurobi.com. [Online]. Available: https://www.gurobi.com/products/gurobi-optimizer/ [Accessed: Aug. 7, 2020].
[17] *pygobnilp manual (version 1.0)*, J. Cussens, University of York, January 30, 2020.
[18] M. Lavielle (2016, Nov. 30). *Maximum likelihood estimation in a Gaussian regression model.* Statistics in Action with R [Online]. Available: http://sia.webpopix.org/regressionML.html. [Accessed: 29 May 2020].
[19] O. C. Carrasco (2019, Jun. 3). *Gaussian mixture models explained.* Towards Data Science [Online]. Available: https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95. [Accessed: Aug. 24, 2020].

[20] H. Akaike, “A new look at the statistical model identification,” *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716-723, December 1974.
[21] G. Schwarz, “Estimating the dimension of a model,” *Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.
[22] I. Tsamardinos, et al., “The max-min hill-climbing Bayesian network structure learning algorithm,” *Machine learning,* vol. 65, no. 1, pp. 31- 78, 2006.