# 3.5-B 推理学习：概率图——贝叶斯网络

概率图（probabilistic graphical model，PGM）是一种概率模型，其中图作为一个数据结构表示随机变量间的条件依赖关系和概率分布，在应用图论的知识解决概率论相关问题同时，且能够可视化随机变量的关系从而易于理解模型结构。概率图已经发展成熟，并广泛应用到各个领域，其中系统的经典论著有*Probabilistic Graphical Models : Principles and Techniques（概率图模型原理与技术）（2009）*[1]。为了避免概率图中繁琐的计算，使用 Python 解决 PGM 问题的库也得以发展，如下表：

| 序号 | Python 库 | 库的基本情况 | 备注 |
| --- | --- | --- | --- |
| 1 | [pgmpy](https://github.com/pgmpy/pgmpy)① | 持续更新，是；使用者 $$ 598人；贡献者 $$ 100人 | 专著，*Mastering Probabilistic Graphical Models Using Python : Master Probabilistic Graphical Models by Learning through Real-World Problems and Illustrative Code Examples in Python*[2] ；[在线手册](https://pgmpy.org/)② |
| 2 | [pomegranate](https://github.com/jmschrei/pomegranate) ③ | 持续更新，是；使用者 $$ 891人；贡献者 $$ 64人 | [在线手册](https://github.com/jmschrei/pomegranate/tree/master/docs/tutorials) ④ |
| 3 | [PGM\_PyLib](https://github.com/jona2510/PGM_PyLib)⑤ | 持续更新，否 | 专著，*Probabilistic Graphical Models : Principles and Applications*[3] |
| 4 | [PyGModels](https://github.com/D-K-E/graphical-models/) ⑥ | 持续更新，否 | [在线手册](https://d-k-e.github.io/graphical-models/html/index.html)⑦ |
| 5 | [pyGMs](https://github.com/ihler/pyGMs) ⑧ | 持续更新，否 | / |
| 6 | [pgmPy](https://github.com/indapa/pgmPy) ⑨ | 持续更新，否 | [课程](https://www.coursera.org/learn/probabilistic-graphical-models)⑩ |
| 7 | [pgm](https://github.com/paulorauber/pgm) ⑪ | 持续更新，否 | / |

从上述 PGM 相关库的比较，目前仍在持续更新的有pgmpy和pomegranate两个库，且使用者和贡献者人数相对较多，具有在线手册或者已经出版的研究专著，因此在下述的 PGM 阐述上主要基于pgmpy实现，并参考 PGM 主要的论著及pgmpy库的手册进行书写。

## 3.5.1 基于概率模型的分类器

对于多维随机变量，且之间相互独立，如果可以由已知样本（特征）和对应的事件（分类）得每一随机变量的概率密度函数（边缘密度函数）或联合分布律，并以观测到的随机变量为条件计算后验分布，则可以用这个后验分布预测新样本的分类。例如对鸢尾花（iris）数据集，分布用[Skearn（scikit-learn）](https://scikit-learn.org/stable/modules/naive_bayes.html)⑫的GaussianNB方法，和自定义高斯朴素贝叶斯分类器阐述连续型随机变量概率模型的构建；并将鸢尾花数据集连续型随机变量转化为离散型随机变量，构建用于离散型随机变量分类的概率模型。

### 3.5.1.1 连续型随机变量分类器

Sklearn的Naive Bayes（朴素贝叶斯）目前集成了”Gaussian Naive Bayes”、“Multinomial Naive Bayes”、“Complement Naive Bayes”、“Bernoulli Naive Bayes”和”Categorical Naive Bayes”等算法。下面应用GaussianNB试验。

#### 1）scikit-learn的GaussianNB

import daft  
from pgmpy.factors.discrete import TabularCPD  
from pgmpy.models import BayesianModel  
  
from sklearn.datasets import load\_iris  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import GaussianNB  
from sklearn import tree  
from sklearn import preprocessing  
  
import numpy as np  
from scipy.special import logsumexp  
import pandas as pd  
from scipy.stats import norm  
  
from sympy.stats import P, E, variance, Die, Normal,density  
from sympy import simplify,Symbol,lambdify

读取鸢尾花数据集，打印查看数据集情况。总共有4个特征列（‘sepal length (cm)’, ‘sepal width (cm)’, ‘petal length (cm)’, ‘petal width (cm)’），每一特征对应一个随机变量，为多维随机变量。数据含3种分类结果（‘setosa’ ‘versicolor’ ‘virginica’），分别对应标签0、1和2。

iris=load\_iris()  
print(f'---feature names:\n{iris.feature\_names};\n---target names{iris.target\_names}\n---data[:3]:\n{iris.data[:3]};\n---target:\n{iris.target}')

---feature names:  
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'];  
---target names['setosa' 'versicolor' 'virginica']  
---data[:3]:  
[[5.1 3.5 1.4 0.2]  
 [4.9 3. 1.4 0.2]  
 [4.7 3.2 1.3 0.2]];  
---target:  
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2  
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
 2 2]

直接使用集成的GaussianNB算法估计概率模型，并用该模型测试数据集75个观测样本，正确率为。

X, y=load\_iris(return\_X\_y=True)  
X\_train, X\_test, y\_train, y\_test=train\_test\_split(X, y, test\_size=0.5, random\_state=0)  
gnb=GaussianNB()  
y\_pred=gnb.fit(X\_train, y\_train).predict(X\_test)  
print("Number of mislabeled points out of a total %d points : %d" % (X\_test.shape[0], (y\_test != y\_pred).sum()))

Number of mislabeled points out of a total 75 points : 4

使用集成封装后的方法不能够直接观察计算的过程，因此提取GaussianNB方法的主要代码行，整理如下。其中update\_mean\_variance方法用于计算用于高斯分布的样本特征均值和方差。joint\_log\_likelihood方法用于构建联合对数概率估计（joint log probability estimates）。

GaussianNB方法位于Sklearn包的naive\_bayes.py模块下。

def update\_mean\_variance(n\_past, mu, var, X):  
 if X.shape[0] == 0:  
 return mu, var  
   
 n\_new=X.shape[0]  
 new\_var=np.var(X, axis=0)  
 new\_mu=np.mean(X, axis=0)  
   
 if n\_past == 0:  
 return new\_mu, new\_var   
   
 n\_total=float(n\_past + n\_new)  
 total\_mu=(n\_new \* new\_mu + n\_past \* mu) / n\_total  
 old\_ssd=n\_past \* var  
 new\_ssd=n\_new \* new\_var  
 total\_ssd=old\_ssd + new\_ssd + (n\_new \* n\_past / n\_total) \* (mu - new\_mu) \*\* 2  
 total\_var=total\_ssd / n\_total  
   
 return total\_mu, total\_var  
  
def joint\_log\_likelihood(X,classes\_,class\_prior\_,var\_,theta\_):  
 joint\_log\_likelihood = []  
 for i in range(np.size(classes\_)):  
 jointi = np.log(class\_prior\_[i])  
 n\_ij = -0.5 \* np.sum(np.log(2.0 \* np.pi \* var\_[i, :]))  
 n\_ij -= 0.5 \* np.sum(((X - theta\_[i, :]) \*\* 2) / (var\_[i, :]), 1)  
 joint\_log\_likelihood.append(jointi + n\_ij)  
   
 joint\_log\_likelihood = np.array(joint\_log\_likelihood).T  
 return joint\_log\_likelihood

X=X\_train  
y=y\_train  
  
\_, n\_features=X.shape  
labelbin=preprocessing.LabelBinarizer()  
Y=labelbin.fit\_transform(y)  
classes\_=labelbin.classes\_  
n\_classes=Y.shape[1]  
class\_log\_prior\_=np.full(n\_classes, -np.log(n\_classes))  
  
unique\_y=np.unique(y)  
  
class\_count\_=np.zeros(n\_classes, dtype=np.float64)  
theta\_=np.zeros((n\_classes, n\_features))  
var\_ = np.zeros((n\_classes, n\_features))  
  
var\_smoothing=1e-9  
epsilon\_=var\_smoothing \* np.var(X, axis=0).max()  
  
for y\_i in unique\_y:  
 i=classes\_.searchsorted(y\_i)  
 X\_i=X[y == y\_i, :]  
 N\_i=X\_i.shape[0]  
   
 new\_theta, new\_sigma=update\_mean\_variance(class\_count\_[i],theta\_[i, :],var\_[i, :],X\_i)  
 theta\_[i, :]=new\_theta  
 var\_[i, :]=new\_sigma  
 class\_count\_[i] += N\_i  
   
var\_[:, :] += epsilon\_  
  
class\_prior\_=class\_count\_ / class\_count\_.sum()  
jll=joint\_log\_likelihood(X,classes\_,class\_prior\_,var\_,theta\_)  
print(jll[:3])  
predicted\_y=classes\_[np.argmax(jll, axis=1)]  
print(predicted\_y)

[[ 0.8762294 -32.51665712 -73.95573104]  
 [-556.7018906 -5.71630183 -3.68412747]  
 [-119.04667384 -5.66092211 -28.96539079]]  
[0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0  
 1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 2 0 1 2 2 0 1 1 2 1 0 0 0 2 1 2  
 0]

#### 2） 自定义高斯朴素贝叶斯分类器

朴素（简单）贝叶斯分类器（Naive Bayes Classifier，NBC）是基于给定类变量（分类）的所有属性（对应特征的随机变量）都是独立的假设，有，及，可以推导出。因为作为常数输入，所以分类规则为：。可以使用最大后验概率（ Maximum A Posteriori，MAP）估计和。不同类的朴素贝叶斯分类器的差异主要在于对的定义[4]。

虽然朴素贝叶斯的假设过于简化，但在现实世界的许多情况下运行良好；且可以用少量的训练数据来估计必要的参数。

定义高斯朴素贝叶斯分类器Gaussian\_naive\_bayes\_classifier类。为了简化代码书写，更清晰的表述分类器的算法机制，对随机变量概率密度函数（使用高斯分布）的定义，使用了SymPy库直接书写；并使用pandas的DataFrame数据结构，尽量一一对应不同变量之间的对应关系，便于理解。

class Gaussian\_naive\_bayes\_classifier:   
 '''  
 高斯朴素贝叶斯分类器。使用到SymPy库计算  
 '''  
  
 def fit(self,X, y,feature\_names):   
 '''  
 由已知样本数据，构建各随机变量边缘概率密集函数，预测新样本类别函数和计算精度  
  
 Parameters  
 ----------  
 X : array  
 样本特征.  
 y : array  
 分类标签.  
 feature\_names : List[string]  
 特征名称列表.  
  
 Returns  
 -------  
 Class  
 类的属性值.  
  
 '''  
 self.feature\_names=feature\_names  
   
 df=pd.DataFrame(X,columns=feature\_names)  
 df['target']=y   
 mean\_std=df.groupby(by=['target']).agg(["mean","std"])   
 self.norms=mean\_std.apply(self.norm\_func,args=(feature\_names,), axis=1, result\_type='expand')   
 joint\_prob=self.job\_prob\_func(df)   
 self.mislabeled(joint\_prob,X,y)  
   
 return self  
   
 def norm\_func(self,row,feature\_names):  
 '''  
 由已知样本数据计算的均值和标注差，构建各随机变量边缘概率密集函数（假设服从正态分布）。用于DataFrame.apply()方法  
  
 Parameters  
 ----------  
 row : Series  
 由已知样本数据计算的均值和标注差.  
 feature\_names : List[string]  
 特征名称列表.  
  
 Returns  
 -------  
 Series  
 随机变量的边缘密度函数.  
  
 '''  
 norms\_lst=[]  
 for feature\_name in feature\_names:  
 dens=density(Normal('X',row[feature\_name]['mean'],row[feature\_name]['std']))  
 norms\_lst.append(dens)  
  
 return pd.Series(norms\_lst,index=feature\_names)   
   
 def prob\_prod(self,row,norms\_df):  
 '''  
 假设随机变量相互独立，由边缘密度函数计算给定特征值的概率。用于DataFrame.apply()方法  
  
 Parameters  
 ----------  
 row : Series  
 特征值.  
 norms\_df : DataFrame  
 随机变量的边缘密度函数.  
  
 Returns  
 -------  
 Series  
 概率值.  
  
 '''  
 p\_prods=[]  
 idxes=[]  
 for idx,norms\_target in norms\_df.iterrows():  
 norms\_target\_lst=norms\_target.values   
 p\_prod=np.prod([norm(x).evalf() for norm,x in zip(norms\_target\_lst,row)])  
 p\_prods.append(p\_prod)  
 idxes.append(idx)  
 return pd.Series(p\_prods,index=idxes)   
   
 def job\_prob\_func(self,df):  
 '''  
 联合概率  
  
 Parameters  
 ----------  
 df : DataFrame  
 特征值.  
  
 Returns  
 -------  
 joint\_prob : DataFrame  
 各个样本对应分类的概率.  
  
 '''  
 joint\_prob=df.apply(self.prob\_prod,args=(self.norms,),axis=1)  
 joint\_prob['p\_max\_target']=joint\_prob.apply(lambda row:row.astype(float).idxmax(),axis=1)   
   
 return joint\_prob  
   
 def predict(self,X):  
 '''  
 根据构建的后验分布（预测模型）预测新样本数据的分类  
  
 Parameters  
 ----------  
 X : array  
 样本特征值.  
  
 Returns  
 -------  
 joint\_prob\_predicted : DataFrame  
 预测分类结果和概率.  
  
 '''  
 df=pd.DataFrame(X,columns=self.feature\_names)  
 joint\_prob\_predicted=self.job\_prob\_func(df)   
   
 return joint\_prob\_predicted  
   
 def mislabeled(self,joint\_prob,X,y):  
 '''  
 计算分类精度  
  
 Parameters  
 ----------  
 joint\_prob : DataFrame  
 预测分类结果数据.  
 X : array  
 样本特征值.  
 y : array  
 样本分类值.  
  
 Returns  
 -------  
 None.  
  
 '''  
 joint\_prob['target']=y  
 joint\_prob['correct']=joint\_prob.apply(lambda row:row.p\_max\_target==row.target,axis=1)   
 print("Number of mislabeled points out of a total %d points : %d" % (X.shape[0],X.shape[0]-joint\_prob['correct'].sum()))   
   
  
iris\_classification=Gaussian\_naive\_bayes\_classifier()  
iris\_classification.fit(X\_train,y\_train,['sepal\_length','sepal\_width','petal\_length','petal\_width'])  
test\_predicted=iris\_classification.predict(X\_test)  
iris\_classification.mislabeled(test\_predicted,X\_test,y\_test)  
test\_predicted.head()

Number of mislabeled points out of a total 75 points : 2  
Number of mislabeled points out of a total 75 points : 5

| **0** | **1** | **2** | **p\_max\_target** | **target** | **correct** |
| --- | --- | --- | --- | --- | --- |
| **0** | 3.99493353970977e-289 | 1.05404275436796e-6 | 0.0449990901096858 | 2 | 2 | True |
| **1** | 4.56660272749500e-102 | 0.119858087756389 | 4.33560630854775e-10 | 1 | 1 | True |
| **2** | 0.312501002072666 | 6.15924326913282e-18 | 1.25771736578052e-32 | 0 | 0 | True |
| **3** | 1.89560807069258e-374 | 6.83814201533865e-7 | 0.136003790622231 | 2 | 2 | True |
| **4** | 14.7330640043679 | 1.04705609546837e-13 | 1.51596858717064e-30 | 0 | 0 | True |

打印随机变量的边缘密度函数，即，由SymPy方式表示。

iris\_classification.norms

|  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** |
| --- | --- | --- | --- | --- |
| **target** |  |  |  |  |
| **0** | NormalDistribution(4.97586206896552, 0.3269503... | NormalDistribution(3.35862068965517, 0.3698621... | NormalDistribution(1.44827586206897, 0.1298919... | NormalDistribution(0.23448275862069, 0.0936401... |
| **1** | NormalDistribution(5.935, 0.5193872908483) | NormalDistribution(2.71, 0.295403381657585) | NormalDistribution(4.185, 0.518372960633355) | NormalDistribution(1.3, 0.220047841688072) |
| **2** | NormalDistribution(6.77692307692308, 0.6358035... | NormalDistribution(3.09230769230769, 0.3248668... | NormalDistribution(5.73461538461538, 0.5705732... | NormalDistribution(2.10769230769231, 0.2225723... |

### 3.5.1.2 离散型随机变量分类器

仍然使用鸢尾花数据集，但是需要将特征值转化为整数形式，构成离散型随机变量的输入。通过self.joint\_prob=Xy.groupby(by=Xy.columns.tolist()).size() / len(Xy)代码计算多维随机变量的联合分布律，结果如iris\_classification\_discrete.joint\_prob所示，并通过已知分类计算后验概率，则可以通过后验概率推断新样本对应的分类值，即将离散型样本特征值对应到联合分布律，结果可能有对应不同分类结果的多个概率值，其中最大概率对应的分类即为估计的分类结果。

从测试数据集的预测结果来看，正确率为。

mini\_iris = np.round(iris.data[:, :]).astype(int)  
data = pd.DataFrame(mini\_iris, columns=['sepal\_length','sepal\_width','petal\_length','petal\_width'])  
data['type'] = iris.target  
  
#Shuffle data  
data = data.iloc[np.random.permutation(len(data))]  
X\_train, X\_test=data[:120], data[120:]  
X\_train

|  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** | **type** |
| --- | --- | --- | --- | --- | --- |
| **86** | 7 | 3 | 5 | 2 | 1 |
| **78** | 6 | 3 | 4 | 2 | 1 |
| **61** | 6 | 3 | 4 | 2 | 1 |
| **89** | 6 | 2 | 4 | 1 | 1 |
| **138** | 6 | 3 | 5 | 2 | 2 |
| **...** | ... | ... | ... | ... | ... |
| **90** | 6 | 3 | 4 | 1 | 1 |
| **38** | 4 | 3 | 1 | 0 | 0 |
| **121** | 6 | 3 | 5 | 2 | 2 |
| **54** | 6 | 3 | 5 | 2 | 1 |
| **84** | 5 | 3 | 4 | 2 | 1 |

120 rows × 5 columns

class Discrete\_probabilisitic\_model\_classifier:  
 '''  
 定义多维离散型随机变量分类器概率模型  
 '''   
 def fit(self,Xy):  
 self.joint\_prob=Xy.groupby(by=Xy.columns.tolist()).size() / len(Xy)  
   
 return self  
   
 def predict(self,X):  
 predicted\_values = []  
 for idx,row in X.iterrows():   
 try:  
 for i,v in enumerate(row):  
 if i==0:  
 multiIndex\_val=self.joint\_prob[v]  
 else:  
 multiIndex\_val=multiIndex\_val[v]  
   
 predicted\_values.append(multiIndex\_val.idxmax())  
 except:  
 predicted\_values.append(-1)  
   
 return predicted\_values  
   
 def mislabeled(self,y\_,y):  
 y\_y=zip(y\_,y)  
 mislabeled\_num=sum([i[0]!=i[1] for i in y\_y])  
 print("Number of mislabeled points out of a total %d points : %d" % (len(y\_),mislabeled\_num))   
  
iris\_classification\_discrete=Discrete\_probabilisitic\_model\_classifier()  
iris\_classification\_discrete.fit(X\_train)  
predicted\_values=iris\_classification\_discrete.predict(X\_test[['sepal\_length','sepal\_width','petal\_length','petal\_width']])  
iris\_classification\_discrete.mislabeled(predicted\_values,X\_test.type.to\_numpy())

Number of mislabeled points out of a total 30 points : 2

iris\_classification\_discrete.joint\_prob

sepal\_length sepal\_width petal\_length petal\_width type  
4 2 1 0 0 0.008333  
 3 1 0 0 0.025000  
5 2 3 1 1 0.025000  
 4 1 1 0.008333  
 2 2 0.008333  
 3 1 0 0 0.041667  
 2 0 0 0.075000  
 4 1 1 0.008333  
 2 1 0.008333  
 4 1 0 0 0.058333  
 2 0 0 0.058333  
 1 0 0.008333  
6 2 4 1 1 0.033333  
 2 1 0.008333  
 5 2 1 0.008333  
 2 0.016667  
 3 4 1 1 0.116667  
 2 1 0.041667  
 5 1 1 0.008333  
 2 1 0.025000  
 2 0.108333  
 6 1 2 0.008333  
 2 2 0.041667  
 4 1 0 0 0.016667  
 2 0 0 0.016667  
7 3 4 1 1 0.016667  
 5 1 1 0.016667  
 2 1 0.025000  
 2 0.025000  
 6 2 2 0.075000  
 4 6 2 2 0.008333  
8 3 6 2 2 0.008333  
 7 2 2 0.025000  
 4 6 2 2 0.008333  
 7 2 2 0.008333  
dtype: float64

## 3.5.2 离散型贝叶斯网络——表征（Representation）与推理（Inference）

从前文基于概率模型构建的分类器可以得知，联合概率分布与每个随机变量的状态（样本点）基数（cardinality）成指数关系，如果每个具有10个状态的10个随机变量，其联合分布的大小将为，因此具有大量特征和状态基数的复杂问题简单的应用概率会由于计算限制，使得推理变得困难。因此，引入概率图模型（Proababilistic Graphical Models，PGM），结合概率论和图论，通过利用随机变量之间的（条件）独立性紧凑的表征联合概率分布的技术。使用图可以简明地表示一组变量之间的依赖关系和独立性，从而能够进行有效的推理。每个图模型的特征在于图结构，例如为有向图、无向图或二者兼有，及图相关联的一组参数，例如对应每个节点（表示随机变量）的（条件）概率分布表（Conditional probability distribution，CPD）。

问题描述：这里参考[InVEST（Integrated Valuation of Ecosystem Services and Tradeoffs）](https://naturalcapitalproject.stanford.edu/software/invest)⑬中生境质量、生境风险评估与碳储存和封存构建了一个假设的简化问题模型，用于阐述 PGM 的贝叶斯网络（Bayesian Network，BN）。这个问题包含5个随机变量，

* 生境威胁（Threat，TH），对栖息地构成威胁的影响大小，假设包含3个状态为：；
* 生境类型（Habitat，HA），即土地利用和覆盖类型（LULC）是否适合生物繁衍，假设包含2个状态为：;
* 碳储存（Carbon Storage，CS），适合生物栖息的生境通常具有良好的自然环境，往往也具有更高的碳储存水平，例如森林在碳库（carbon pools）中含有最多的碳，假设包含3个状态为：；
* 生境质量（Habitat Quality，HQ），为生物多样性的一种反映，假设包含2个状态为：}；
* 生境风险（Habitat Risk，HR），评估人类活动对生境和物种造成的累积风险，假设包含2个状态为：。

那么问题是，如果已知任何一个或多个随机变量的状态，如何估计其它随机变量的状态概率从而推断出最可能的状态，例如，已知生境威胁为'weak'，生境类型为'suitable'，那么如何估计生境质量的状态概率，即为'good'和'bad'的概率，从而推断对应最大概率的状态即为在给定条件下最可能的状态；又或者，已知生境质量和生境风险评估，估计生境威胁或者生境类型状态等。

### 3.5.2.1 表征

PGM 以图为结构，以 CPDs为节点（随机变量）参数，因此为了以一种易于图示理解的方式构建 PGM，

1. 首先用[daft](https://docs.daft-pgm.org/en/latest/)⑭ 库根据问题设计构建图，表达随机变量和随机变量之间的关系影响；
2. 根据图，使用[pgmpy](https://pgmpy.org/index.html)①库建立各个随机变量的 CPDs；
3. 将daft设计的图转化为pgmpy库的图模型结构，并将 CPDs 链接到图上，完成 PGM的构建。

使用daft库的目的是因为目前pgmpy库还未包含打印可视化图的方法，但是可以直接用该库提供的BayesianNetwork等方法构建图模型（不能直接可视化图，可通过to\_daft方法转化为daft库支持的数据结构后可视化）。构建完 BN（PGM） 后就可以用pgmpy库提供的各类方法进行推断与学习。

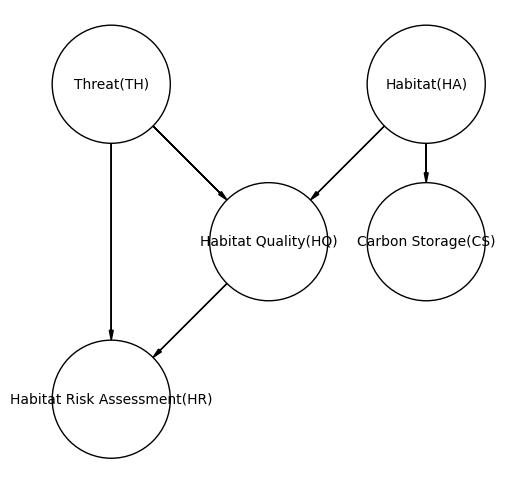
# IPython extension to reload modules before executing user code.  
%load\_ext autoreload   
# Reload all modules (except those excluded by %aimport) every time before executing the Python code typed.  
%autoreload 2   
import usda.pgm as usda\_pgm  
  
import daft  
from pgmpy.models import BayesianNetwork  
from pgmpy.factors.discrete import TabularCPD  
from pgmpy.inference import VariableElimination,BeliefPropagation  
  
import copy  
import matplotlib.pyplot as plt

#### 1）定义图

BN 将个（离散型）随机变量的联合分布表示为有向无环图（Directed Acyclic Graph，DAG）和一组（条件）概率分布（CPDs）。图中的节点（node/vertice，顶点）代表随机变量，例如上述问题描述中的生境威胁（TH）、生境类型（HA）、碳储存（CS）、生境质量（HQ）和生境风险（HR），可记作。边（edge）代表随机变量间的相关性（dependencies）(直接影响，节点及其邻接父节点的依赖关系)，例如生境质量（HQ）依赖于生境威胁（TH）和生境类型（HA），因此构建有向边和；碳储存（CS）依赖于生境类型（HA），因此有；生境风险（HR）依赖于生境威胁（TH）和生境质量（HQ），因此有和，上述所有边可记作。

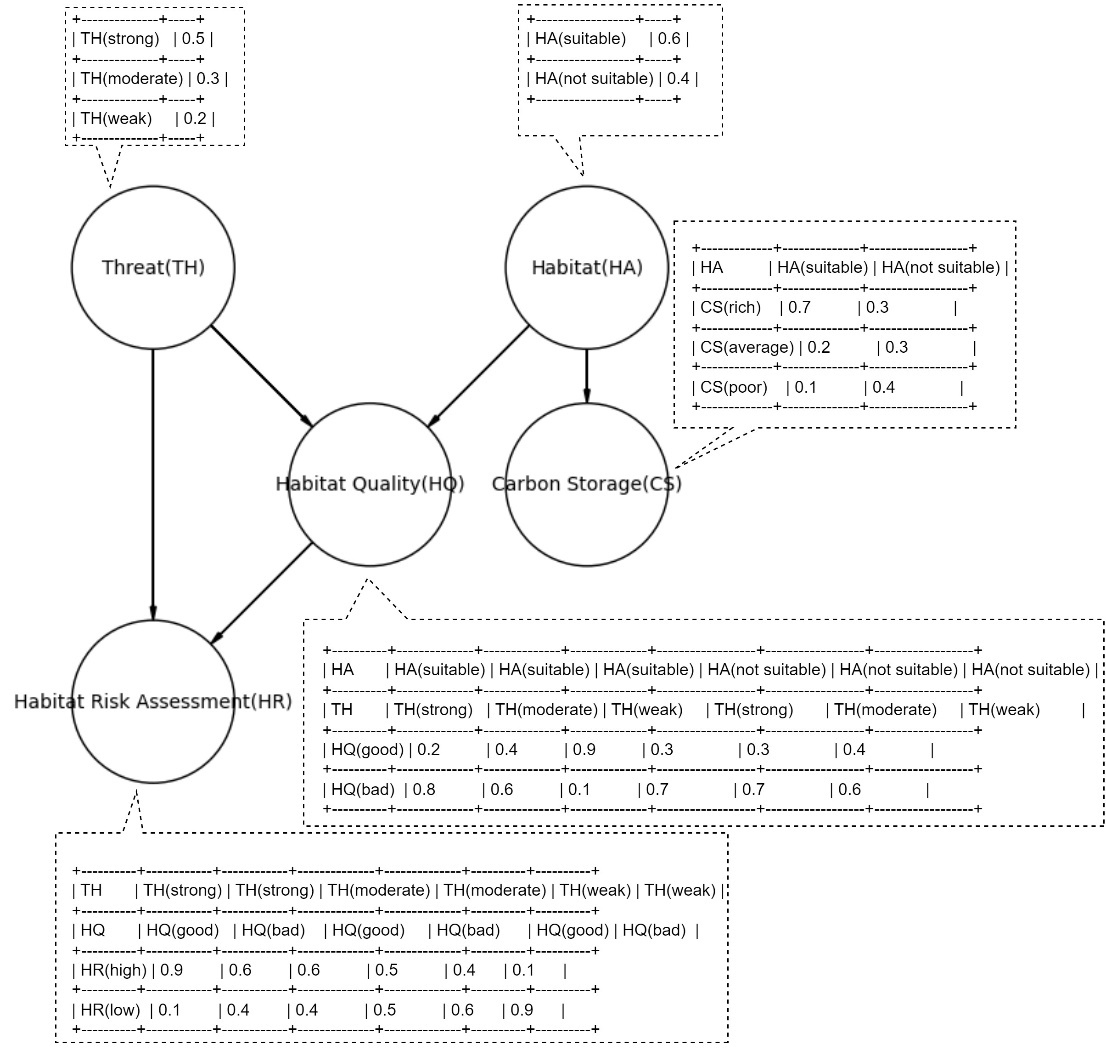
图以因式分解（Factorization）的方式紧凑的表示了随机变量联合分布的数据结构骨架；也可理解为关于随机变量联合分布的一系列条件独立性假设的紧凑表示。

pgm\_habitat=daft.PGM(grid\_unit=4.0, node\_unit=3)  
  
pgm\_habitat.add\_node(daft.Node('HA', r"Habitat(HA)", 3, 3))  
pgm\_habitat.add\_node(daft.Node('TH', r"Threat(TH)", 1, 3))  
pgm\_habitat.add\_node(daft.Node('CS', r"Carbon Storage(CS)", 3, 2))  
pgm\_habitat.add\_node(daft.Node('HQ', r"Habitat Quality(HQ)", 2, 2))  
pgm\_habitat.add\_node(daft.Node('HR', r"Habitat Risk Assessment(HR)", 1, 1))  
  
pgm\_habitat.add\_edge('TH','HQ')  
pgm\_habitat.add\_edge('HA','HQ')  
pgm\_habitat.add\_edge('HA','CS')  
pgm\_habitat.add\_edge('TH','HR')  
pgm\_habitat.add\_edge('HQ','HR')  
  
pgm\_habitat.render();



#### 2）因子分解

一般来讲，BN 中的每一个随机变量都关联着一个 CPD，在已知其父节点每种可能状态的条件下，用来指明状态的一个分布，记为，式中，为图中节点的父节点，例如随机变量依赖于和，其 CPD 表示为，对应着一个条件分布律（表）为变量cpd\_hq\_sn。对没有父节点的节点，例如和，其CPD以空集为条件，实际上即为边缘分布和。对应各随机变量节点的 CPDs 图如下：



对于上述 BN（PGM）所有随机变量（属性、特征）的联合概率分布，考虑到随机变量之间的相互独立性，可以得到，即对于 BN 的随机变量有联合概率分布，该公式为**贝叶斯网的链式法则**。可以表述为基于随机变量的独立性假设，PGM 分布根据图的因子分解（factorization），对应公式为。

* 定义参数：条件概率分布

使用pgmpy库的TabularCPD方法建立随机变量的 CPDs。

cpd\_ha\_sn=TabularCPD(variable='HA', variable\_card=2, values=[[0.6], [0.4]], state\_names={'HA': ['suitable', 'not suitable']})  
cpd\_th\_sn=TabularCPD(variable='TH', variable\_card=3, values=[[0.5], [0.3],[0.2]], state\_names={'TH': ['strong', 'moderate','weak']})  
  
cpd\_cs\_sn=TabularCPD(variable='CS', variable\_card=3,   
 values=[[0.7,0.3],  
 [0.2,0.3],  
 [0.1,0.4]],  
 evidence=['HA'],  
 evidence\_card=[2],  
 state\_names={'CS': ['rich', 'average','poor'],  
 'HA': ['suitable', 'not suitable']})  
  
cpd\_hq\_sn=TabularCPD(variable='HQ', variable\_card=2,   
 values=[[0.2,0.4,0.9,0.3,0.3,0.4],  
 [0.8,0.6,0.1,0.7,0.7,0.6]],  
 evidence=['HA', 'TH'],  
 evidence\_card=[2, 3],  
 state\_names={'HQ': ['good', 'bad'],  
 'HA': ['suitable', 'not suitable'],  
 'TH': ['strong', 'moderate','weak']})  
cpd\_hr\_sn=TabularCPD(variable='HR', variable\_card=2,  
 values=[[0.9,0.6,0.6,0.5,0.4,0.1],  
 [0.1,0.4,0.4,0.5,0.6,0.9]],  
 evidence=['TH','HQ'],  
 evidence\_card=[3,2],  
 state\_names={'HR': ['high', 'low'],   
 'TH': ['strong', 'moderate','weak'],  
 'HQ': ['good', 'bad']})   
  
  
print(f'# Habitat\n{cpd\_ha\_sn}')  
print(f'# Threat\n{cpd\_th\_sn}')  
print(f'# Carbon Storage\n{cpd\_cs\_sn}')  
print(f'# Habitat Quality\n{cpd\_hq\_sn}')  
print(f'# Habitat Risk Assessment\n{cpd\_hr\_sn}')

# Habitat  
+------------------+-----+  
| HA(suitable) | 0.6 |  
+------------------+-----+  
| HA(not suitable) | 0.4 |  
+------------------+-----+  
# Threat  
+--------------+-----+  
| TH(strong) | 0.5 |  
+--------------+-----+  
| TH(moderate) | 0.3 |  
+--------------+-----+  
| TH(weak) | 0.2 |  
+--------------+-----+  
# Carbon Storage  
+-------------+--------------+------------------+  
| HA | HA(suitable) | HA(not suitable) |  
+-------------+--------------+------------------+  
| CS(rich) | 0.7 | 0.3 |  
+-------------+--------------+------------------+  
| CS(average) | 0.2 | 0.3 |  
+-------------+--------------+------------------+  
| CS(poor) | 0.1 | 0.4 |  
+-------------+--------------+------------------+  
# Habitat Quality  
+----------+--------------+--------------+-----+------------------+------------------+  
| HA | HA(suitable) | HA(suitable) | ... | HA(not suitable) | HA(not suitable) |  
+----------+--------------+--------------+-----+------------------+------------------+  
| TH | TH(strong) | TH(moderate) | ... | TH(moderate) | TH(weak) |  
+----------+--------------+--------------+-----+------------------+------------------+  
| HQ(good) | 0.2 | 0.4 | ... | 0.3 | 0.4 |  
+----------+--------------+--------------+-----+------------------+------------------+  
| HQ(bad) | 0.8 | 0.6 | ... | 0.7 | 0.6 |  
+----------+--------------+--------------+-----+------------------+------------------+  
# Habitat Risk Assessment  
+----------+------------+------------+--------------+--------------+----------+----------+  
| TH | TH(strong) | TH(strong) | TH(moderate) | TH(moderate) | TH(weak) | TH(weak) |  
+----------+------------+------------+--------------+--------------+----------+----------+  
| HQ | HQ(good) | HQ(bad) | HQ(good) | HQ(bad) | HQ(good) | HQ(bad) |  
+----------+------------+------------+--------------+--------------+----------+----------+  
| HR(high) | 0.9 | 0.6 | 0.6 | 0.5 | 0.4 | 0.1 |  
+----------+------------+------------+--------------+--------------+----------+----------+  
| HR(low) | 0.1 | 0.4 | 0.4 | 0.5 | 0.6 | 0.9 |  
+----------+------------+------------+--------------+--------------+----------+----------+

* 链接图和条件概率分布

转化daft库构建的图为pgmpy库的形式，并将pgmpy库建立的 CPDs 链接到该图上，完成 BN（PGM）的构建。可以通过check\_mode方法检验构建的 PGM 是否正确，如果提示随机变量的边缘分布不为1等情况，则需要对假设的概率数值进行调整，直至检查返回True，表明构建的 PGM 成立。

bn\_habitat=usda\_pgm.convert\_pgm\_to\_pgmpy(pgm\_habitat)  
bn\_habitat.add\_cpds(cpd\_ha\_sn,cpd\_th\_sn,cpd\_cs\_sn,cpd\_hq\_sn,cpd\_hr\_sn)  
bn\_habitat.check\_model()

True

建立 PGM 后可以读取计算相关信息，例如随机变量的 CPDs，状态基数等。

print(bn\_habitat.get\_cpds())  
print(bn\_habitat.get\_cpds('HA'))  
print(bn\_habitat.get\_cardinality('TH'))

[<TabularCPD representing P(HA:2) at 0x10bcceb5970>, <TabularCPD representing P(TH:3) at 0x10bf64e1a30>, <TabularCPD representing P(CS:3 | HA:2) at 0x10bcceb59a0>, <TabularCPD representing P(HQ:2 | HA:2, TH:3) at 0x10bf64e1d00>, <TabularCPD representing P(HR:2 | TH:3, HQ:2) at 0x10bcceb5a00>]  
+------------------+-----+  
| HA(suitable) | 0.6 |  
+------------------+-----+  
| HA(not suitable) | 0.4 |  
+------------------+-----+  
3

#### 3）图中的独立性

依赖性（Dependencies）和独立性（independencies）是分布的两个主要性质，对理解分布的特征非常重要，并可从根本上降低推断的计算成本。BN 网络结构隐含的独立性可以分为两类：

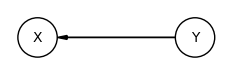
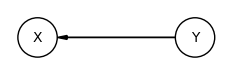
**局部独立性（Local Independencies）**：图中任何变量都独立于给定其父类的非后代，公式可写为；

**全局独立性（Global Independencies）**：为讨论 BN 中的全局独立性，需要查看各种可能的网络结构，如下内容。

* D-分离（D-separation）

**直接连接（Direct connection）**：当任意图（网络结构）的随机变量（节点）和有边（如下图）时，不论图中其它随机变量的证据（ evidence）如何，都可以构建和相互关联的分布。例如，生境威胁（）和生境风险（）为，如果增加威胁的程度（概率），例如为strong（0.5），那么生境风险评估变高的概率也会增加。

pgm\_=daft.PGM()  
pgm\_.add\_node(daft.Node('X', r"X", 1, 1))  
pgm\_.add\_node(daft.Node('Y', r"Y", 3, 1))  
  
pgm\_xy=copy.deepcopy(pgm\_)  
pgm\_xy.add\_edge('X', 'Y')  
pgm\_xy.render();  
  
#----------------------------------------------  
pgm\_yx=copy.deepcopy(pgm\_)  
pgm\_yx.add\_edge('Y', 'X')  
pgm\_yx.render();



在给定为状态weak和strong时，状态概率有相应变化。

infer\_habitat=VariableElimination(bn\_habitat)  
  
print(infer\_habitat.query(['HR'], evidence={'TH': 'weak'}))  
print('/'\*30)  
print(infer\_habitat.query(['HR'], evidence={'TH': 'strong'}))

+----------+-----------+  
| HR | phi(HR) |  
+==========+===========+  
| HR(high) | 0.3100 |  
+----------+-----------+  
| HR(low) | 0.6900 |  
+----------+-----------+  
//////////////////////////////  
+----------+-----------+  
| HR | phi(HR) |  
+==========+===========+  
| HR(high) | 0.6720 |  
+----------+-----------+  
| HR(low) | 0.3280 |  
+----------+-----------+

在给定为状态high和low时，状态概率也有相应变化。

print(infer\_habitat.query(['TH'], evidence={'HR': 'high'}))  
print('/'\*30)  
print(infer\_habitat.query(['TH'], evidence={'HR': 'low'}))

+--------------+-----------+  
| TH | phi(TH) |  
+==============+===========+  
| TH(strong) | 0.6013 |  
+--------------+-----------+  
| TH(moderate) | 0.2878 |  
+--------------+-----------+  
| TH(weak) | 0.1110 |  
+--------------+-----------+  
//////////////////////////////  
+--------------+-----------+  
| TH | phi(TH) |  
+==============+===========+  
| TH(strong) | 0.3717 |  
+--------------+-----------+  
| TH(moderate) | 0.3155 |  
+--------------+-----------+  
| TH(weak) | 0.3128 |  
+--------------+-----------+

当图中一个影响（influence）能够沿迹（trail）从一个节点流动到另一个节点，称为有效迹（active trail），可以使用is\_dconnected方法判断，例如，对于直接连接的和，其判断结果为True，为有效迹。

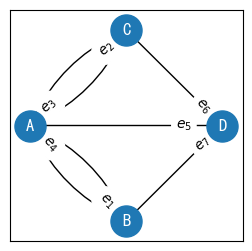
print(bn\_habitat.is\_dconnected('HR','TH'))  
print(bn\_habitat.is\_dconnected('TH','HR'))

True  
True

（补充）途径（Walk）、路（Paths）和迹（Trails）

图中长度为的**途径**是一个非空的顶点（节点）和边交错序列，使得对于所有均有。若，则称此途径是**闭合的（closed）**。如果途径中的顶点互不相同（除收尾顶点），则为中的一条**路**；如果途径中的边互不相同，则为中的一条**迹**。例如下图的哥尼斯堡七桥（Seven Bridges of Königsberg）中为一个路，而则为一个迹。

import networkx as nx  
plt.figure(1,figsize=(3,3))   
  
G = nx.DiGraph()  
G.add\_edge("A", "B", label='$e\_1$')  
G.add\_edge("B", "A", label='$e\_4$')  
G.add\_edge("A", "C", label='$e\_2$')  
G.add\_edge("C", "A", label='$e\_3$')  
G.add\_edge("A", "D", label='$e\_5$')  
G.add\_edge("B", "D", label='$e\_7$')  
G.add\_edge("C", "D", label='$e\_6$')  
  
positions = {"A": (0, 0), "B": (1, -2), "C": (1, 2), "D": (2, 0)}  
  
edge\_labels = nx.get\_edge\_attributes(G, "label")  
  
nx.draw\_networkx\_nodes(G, pos=positions, node\_size=500)  
nx.draw\_networkx\_labels(G, pos=positions, font\_color="w")  
nx.draw\_networkx\_edges(  
 G, pos=positions, edgelist=[("A", "D"), ("B", "D"), ("C", "D")], arrowstyle="-"  
)  
nx.draw\_networkx\_edges(  
 G,  
 pos=positions,  
 edgelist=[("A", "B"), ("B", "A"), ("C", "A"), ("A", "C")],  
 arrowstyle="-",  
 connectionstyle="arc3,rad=0.2",  
)  
nx.draw\_networkx\_edge\_labels(G, pos=positions, edge\_labels=edge\_labels, label\_pos=0.2);



**间接连接（Indirect connection）**：如果与不直接相连，但它们之间在图中存在一条迹，例如存在一条经过的迹，有如下4种情况，因果链（因果作用和证据作用），共同的原因和共同的租用。

1. 间接的因果作用（Indirect causal effect）（图 Causal）：以上述生境问题为例，对于迹，如果观察到生境类型（）适合生物生存，为状态suitable，那么倾向于相信生境质量（）状态good的概率会提高，生境风险（HR）的等级随之会相应提升，状态high的概率增加。即，后续事件的概率对于生境类型的观测值有较强的依赖，可以相信通过可以作用于。但是，如果观察到，即观察到生境质量，那么生境类型不再对生境风险产生影响，为，因此如果观察到，那么不会通过作用于。

pgm\_=daft.PGM()  
  
pgm\_.add\_node(daft.Node('X', r"X", 1, 3))  
pgm\_.add\_node(daft.Node('Z', r"Z", 1, 2))  
pgm\_.add\_node(daft.Node('Y', r"Y", 1, 1))  
  
#----------------------------------------------  
pgm\_causal=copy.deepcopy(pgm\_)  
pgm\_causal.add\_edge('X', 'Z')  
pgm\_causal.add\_edge('Z', 'Y')  
pgm\_causal.add\_text(0.7,3.5,label='Causal')  
pgm\_causal.render();

观测到状态为suitable，推断的概率。

print(infer\_habitat.query(['HR'], evidence={'HA': 'suitable'}))

+----------+-----------+  
| HR | phi(HR) |  
+==========+===========+  
| HR(high) | 0.5660 |  
+----------+-----------+  
| HR(low) | 0.4340 |  
+----------+-----------+

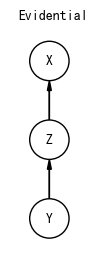
观测到的状态为bad，推断的概率。

print(infer\_habitat.query(['HR'], evidence={'HQ': 'bad'}))

+----------+-----------+  
| HR | phi(HR) |  
+==========+===========+  
| HR(high) | 0.5222 |  
+----------+-----------+  
| HR(low) | 0.4778 |  
+----------+-----------+

1. 间接的证据作用（Indirect evidential effect）(图 Evidential)：仍然对于迹，如果观察到状态为high，那么相信的suitable状态的概率会增加。同样，一旦观察到，将不会对提高额外的信息。因此，对于情形的分析与关于可以通过来影响这一因果情形的分析一致，但仅限于没有观测到的情况。即可以通过为提供额外的信息（影响），但仅限于没有观测到的情形。因为独立性是一个对称的概念，因此如果不成立，那么也不成立。

#----------------------------------------------  
pgm\_evidential=copy.deepcopy(pgm\_)  
pgm\_evidential.add\_edge('Y', 'Z')  
pgm\_evidential.add\_edge('Z', 'X')  
pgm\_evidential.add\_text(0.6,3.5,label='Evidential')  
pgm\_evidential.render();



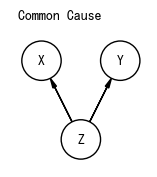
观测到的状态为high，推断的概率。

print(infer\_habitat.query(['HA'], evidence={'HR': 'high'}))

+------------------+-----------+  
| HA | phi(HA) |  
+==================+===========+  
| HA(suitable) | 0.6077 |  
+------------------+-----------+  
| HA(not suitable) | 0.3923 |  
+------------------+-----------+

1. 共同的原因（Common cause）（图Common Cause）：以上述生境问题为例，对于迹，生境类型是生境质量和碳储存的父节点。和在这个模型中相互关联，如果观察到生境质量较好，则提供了有关生境类型的信息，那么可以推测较为适合生物栖息，具有较好的生态系统，例如茂密的森林，进一步可以辅助推断碳储存的状态概率，趋向于较为丰富的碳储量。然而，一旦观察到，和之间的关联就会消失，不再提供有关的任何信息。即对于迹，可以通过影响，但仅限于没有观测到的情形。

#----------------------------------------------  
pgm\_=daft.PGM()  
  
pgm\_.add\_node(daft.Node('X', r"X", 1, 2))  
pgm\_.add\_node(daft.Node('Z', r"Z", 1.5, 1))  
pgm\_.add\_node(daft.Node('Y', r"Y", 2, 2))  
  
pgm\_common\_cause=copy.deepcopy(pgm\_)  
pgm\_common\_cause.add\_edge('Z', 'X')  
pgm\_common\_cause.add\_edge('Z', 'Y')  
pgm\_common\_cause.add\_text(0.7,2.5,label='Common Cause')  
pgm\_common\_cause.render();



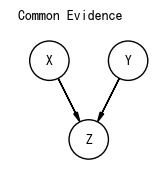
观测到的状态为good，推断的概率。

print(infer\_habitat.query(['CS'], evidence={'HQ': 'good'}))

+-------------+-----------+  
| CS | phi(CS) |  
+=============+===========+  
| CS(rich) | 0.5609 |  
+-------------+-----------+  
| CS(average) | 0.2348 |  
+-------------+-----------+  
| CS(poor) | 0.2043 |  
+-------------+-----------+

1. 共同的作用（Common effect ）(图Common Evidence)：对于上述3种情况，可以通过来影响当且仅当没有观测到。但是，对于迹，例如对于上述生境问题，有，和都是的父节点，当没有观测到时，根据网络结构的局部独立性，和相互独立。因此，如果没有观测到，影响将无法沿着迹流动。

#----------------------------------------------  
pgm\_common\_evidence=copy.deepcopy(pgm\_)  
pgm\_common\_evidence.add\_edge('X', 'Z')  
pgm\_common\_evidence.add\_edge('Y', 'Z')  
pgm\_common\_evidence.add\_text(0.6,2.5,label='Common Evidence')  
pgm\_common\_evidence.render();



通过is\_dconnected方法得出和并不是有效迹，如果观测到状态为strong，可以发现的状态概率并没有发生变化；但是如果观测到，可以发现此时和转变成有效迹，当同时观测到HQ的状态为good时，估计的状态概率发生了变化。

print(bn\_habitat.is\_dconnected('HA','TH'))  
print(infer\_habitat.query(['HA'], evidence={'TH':'strong'}))  
print(bn\_habitat.is\_dconnected('HA','TH',observed='HQ'))  
print(infer\_habitat.query(['HA'], evidence={'TH':'strong','HQ': 'good'}))

False  
+------------------+-----------+  
| HA | phi(HA) |  
+==================+===========+  
| HA(suitable) | 0.6000 |  
+------------------+-----------+  
| HA(not suitable) | 0.4000 |  
+------------------+-----------+  
True  
+------------------+-----------+  
| HA | phi(HA) |  
+==================+===========+  
| HA(suitable) | 0.5000 |  
+------------------+-----------+  
| HA(not suitable) | 0.5000 |  
+------------------+-----------+

当影响经过可以从流向时，迹是有效的（active）。对有效双边迹的分析结果总结如下：

* **因果迹**：有效当且仅当没有观测到；
* **证据迹**：有效当且仅当没有观测到；
* **共同的原因**：有效当且仅当没有观测到；
* **共同的作用**：有效当且仅当观察到或者的一个后代。形如的结构称为一个**v-结构（v-structure）**。

将概率影响看作图中的流动非常有用，从上述网络结构可知，何时影响可以从流经，然后影响到关于的置信度（beliefs）。

推演到一般情况，考虑迹比较长的情形，。直观上，对于从流动到的影响，需要流经这条迹上的每个单元点。即，如果每条双边迹都允许影响流过，那么可以影响到。

定义：另是一个贝叶斯网结构，且是中的一条迹。另是观测变量的一个子集，在给定的条件下，假如，

* 一旦有一个v-结构，则或其一个后代在中；
* 迹上的其它节点都不在中，

那么迹是**有效迹**。

如果或在中，那么迹不是有效迹。

**d-分离（d-separation）** 提供在有向图节点之间分离的一个概念。

定义：令是图的三个节点集。在给定的条件下，加入任意节点与之间不存在有效迹，那么与在给定时是**d-分离**的，记作。与d-分离相对应的独立性的集合用表示：。这个集合称为**全局马尔可夫独立性（global Markov independencies）**。

local\_independencies方法返回随机变量的独立性实例，包含每个随机变量的局部独立性。例如下述结果中，给定时，和相互独立。

rvs\_habitat=['HA','TH','CS','HQ','HR']  
print(bn\_habitat.local\_independencies(rvs\_habitat))

(HA ⟂ TH)  
(TH ⟂ CS, HA)  
(CS ⟂ TH, HQ, HR | HA)  
(HQ ⟂ CS | TH, HA)  
(HR ⟂ CS, HA | TH, HQ)

get\_independencies方法通过检查d-分离计算有向无环图（ Directed Acyclic Graph，DAG）中的独立性。

print(bn\_habitat.get\_independencies())

(TH ⟂ CS, HA)  
(TH ⟂ HA | CS)  
(TH ⟂ CS | HA)  
(TH ⟂ CS | HR, HA)  
(TH ⟂ CS | HQ, HA)  
(TH ⟂ CS | HR, HQ, HA)  
(HR ⟂ CS | HA)  
(HR ⟂ CS, HA | TH, HQ)  
(HR ⟂ CS | TH, HA)  
(HR ⟂ CS | HQ, HA)  
(HR ⟂ HA | TH, CS, HQ)  
(HR ⟂ CS | TH, HQ, HA)  
(HA ⟂ TH)  
(HA ⟂ TH | CS)  
(HA ⟂ HR | TH, HQ)  
(HA ⟂ HR | TH, CS, HQ)  
(CS ⟂ TH)  
(CS ⟂ TH, HQ, HR | HA)  
(CS ⟂ HR | TH, HQ)  
(CS ⟂ HR, HQ | TH, HA)  
(CS ⟂ TH, HR | HQ, HA)  
(CS ⟂ TH, HQ | HR, HA)  
(CS ⟂ HR | TH, HQ, HA)  
(CS ⟂ HQ | TH, HA, HR)  
(CS ⟂ TH | HR, HQ, HA)  
(HQ ⟂ CS | HA)  
(HQ ⟂ CS | TH, HA)  
(HQ ⟂ CS | HR, HA)  
(HQ ⟂ CS | TH, HA, HR)

对于网络结构中的任意两个变量和，如果的任何变化影响的值，那么说和之间存在有效迹。pgmpy库提供的active\_trail\_nodes方法，根据提供的节点返回一组受该节点变化影响的节点集合。例如，当未有观测到的节点状态时，的任何变化将会影响到等节点；如果观测到，那么的任何变化将不会影响到其它任何网络结构中的节点。

print(bn\_habitat.active\_trail\_nodes('CS'))  
print(bn\_habitat.active\_trail\_nodes('CS',observed='HA'))

{'CS': {'CS', 'HQ', 'HA', 'HR'}}  
{'CS': {'CS'}}

### 3.5.2.2 推理

在阐述*图中的独立性*时，已经使用了infer\_habitat=VariableElimination(bn\_habitat)语句和infer\_habitat.query()方法实现了相关推理，条件概率的概率查询（conditional probability query）。条件概率查询适用于许多有用的推理模型，包括解释、预测、因果性推理等。根据条件概率的定义，式中，分子的每个实例是一个概率表达式，可以通过对联合分布中的所有表值求和计算，这些表值对应于的一致赋值。即，令  
W=X−Y−E*,,* P(e)**。

上述在给定联合分布时找到随机变量的条件分布计算中，会出现一个指数级的大表，并已证明，对于推理任务，指数爆炸（exponential blowup）问题在最坏情况下几乎肯定是不可避免地。然而在现实生活中，并不总是遇到最坏的情况，通过用于推理任务的的各种算法可以减小计算量，例如pgmpy库提供的VariableElimination变量消除算法，BeliefPropagation置信传播算法等。

下述事例使用 Daphne Koller 给出的学生示例模型[1]，结合pgmpy提供基于 Python 的推理算法，重述学生示例模型推理模式部分。

直接使用pgmpy构建 BN 模型，通过to\_daft方法转化为daft库的图结构后打印查看图。该学生 BN 模型包含有5个随机变量，

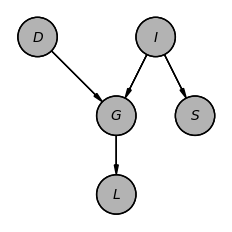
1. 学生的智商（），状态值有；
2. 课程的难度（），状态值有；
3. 考试的成绩（），状态值有；
4. 学生的SAT成绩（）状态值有；
5. 推荐信的质量（）状态值有。

model=BayesianNetwork([('D', 'G'), ('I', 'G'), ('G', 'L'), ('I', 'S')])  
  
cpd\_d\_sn = TabularCPD(variable='D', variable\_card=2, values=[[0.6], [0.4]], state\_names={'D': ['Easy', 'Hard']})  
cpd\_i\_sn = TabularCPD(variable='I', variable\_card=2, values=[[0.7], [0.3]], state\_names={'I': ['Dumb', 'Intelligent']})  
cpd\_g\_sn = TabularCPD(variable='G', variable\_card=3,   
 values=[[0.3, 0.05, 0.9, 0.5],  
 [0.4, 0.25, 0.08, 0.3],  
 [0.3, 0.7, 0.02, 0.2]],  
 evidence=['I', 'D'],  
 evidence\_card=[2, 2],  
 state\_names={'G': ['A', 'B', 'C'],  
 'I': ['Dumb', 'Intelligent'],  
 'D': ['Easy', 'Hard']})  
  
cpd\_l\_sn = TabularCPD(variable='L', variable\_card=2,   
 values=[[0.1, 0.4, 0.99],  
 [0.9, 0.6, 0.01]],  
 evidence=['G'],  
 evidence\_card=[3],  
 state\_names={'L': ['Bad', 'Good'],  
 'G': ['A', 'B', 'C']})  
  
cpd\_s\_sn = TabularCPD(variable='S', variable\_card=2,  
 values=[[0.95, 0.2],  
 [0.05, 0.8]],  
 evidence=['I'],  
 evidence\_card=[2],  
 state\_names={'S': ['Bad', 'Good'],  
 'I': ['Dumb', 'Intelligent']})  
  
model.add\_cpds(cpd\_d\_sn, cpd\_i\_sn, cpd\_g\_sn, cpd\_l\_sn, cpd\_s\_sn)  
model.check\_model()

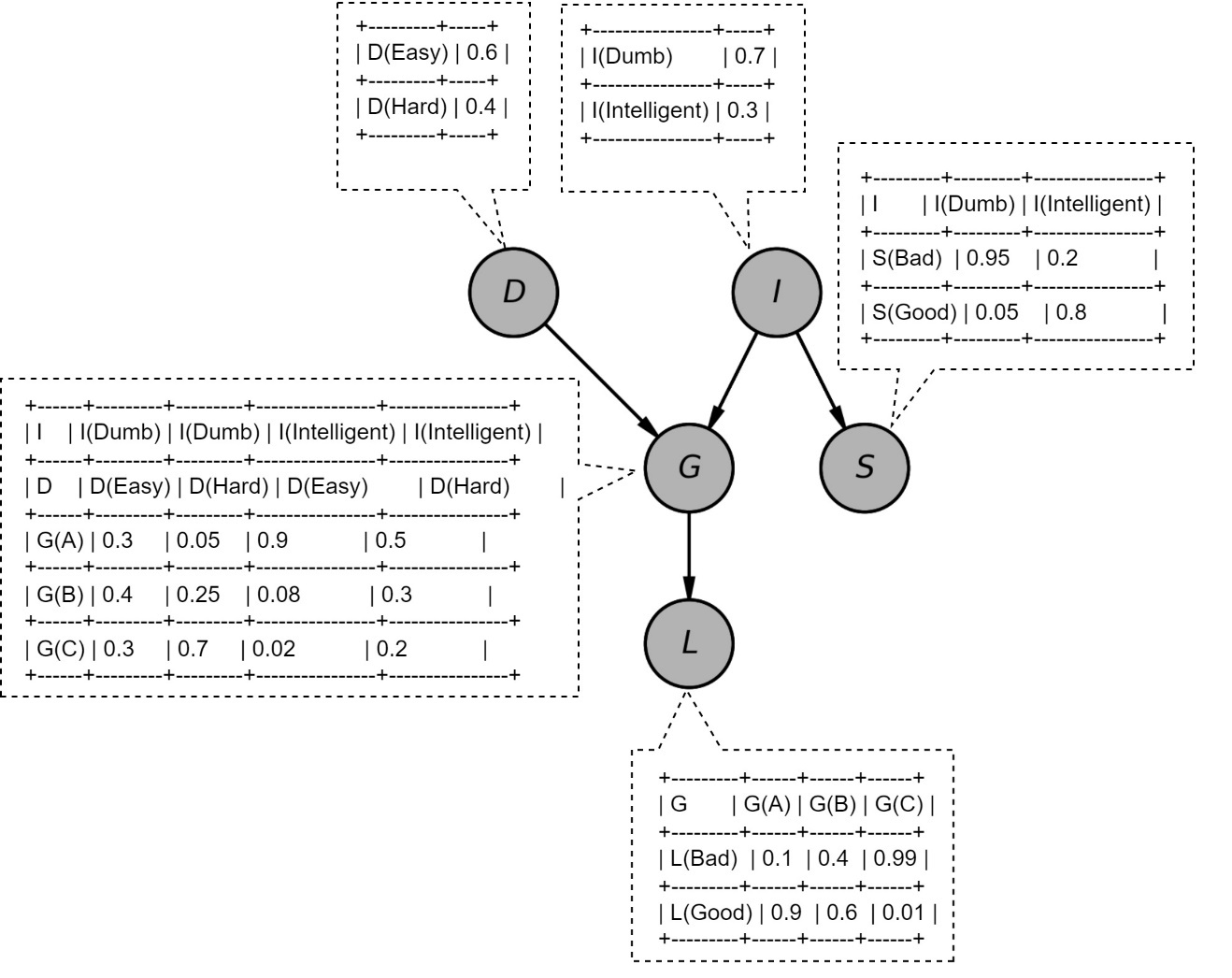
True

转化为daft图结构，打印查看。

model\_daft=model.to\_daft(node\_pos={'D':(1,3),'I':(2.5,3),'G':(2,2),'S':(3,2),'L':(2,1)})  
model\_daft.render(dpi=100);



将 CPDs 和图的随机变量节点对应起来，方便查看。



在给定任意观测值时，联合分布具体指定了任意事件的概率：关于事件，通过在联合分布上消除与观测不一致的表值，调节事件的联合分布，并将结果表值重新归一化，使其总和为1。事件的概率可以通过累加与一致的结果后验分布中的所有表值的概率得到。

例如以学生 George 为例，用上述模型对 George 推理。如果要询问 George 有多大可能从课程 Econ101 的教授那里获得一封好的推荐信。在对 George 和 Econ101 其它情况一无所知的条件下，这一概率为。

infer\_student=VariableElimination(model)  
print(infer\_student.query(['L']))

+---------+----------+  
| L | phi(L) |  
+=========+==========+  
| L(Bad) | 0.4977 |  
+---------+----------+  
| L(Good) | 0.5023 |  
+---------+----------+

现在了解到 George 不太聪明，他从教课程 Econ101 的教授那里获得一封好的推荐信的概率降至到大约。

print(infer\_student.query(['L'],evidence={'I':'Dumb'}))

+---------+----------+  
| L | phi(L) |  
+=========+==========+  
| L(Bad) | 0.6114 |  
+---------+----------+  
| L(Good) | 0.3886 |  
+---------+----------+

进一步得知 Econ101 是一门简单的课程，此时 George 从教授那里获得一封好的推荐信的概率为。这类可以预测已知各种因素（例如， George的智商）对“下游”的影响实例，为**因果推理（causal reasoning）或预测（prediction）**。

print(infer\_student.query(['L'],evidence={'I':'Dumb','D':'Easy'}))

+---------+----------+  
| L | phi(L) |  
+=========+==========+  
| L(Bad) | 0.4870 |  
+---------+----------+  
| L(Good) | 0.5130 |  
+---------+----------+

现在考虑 Acme 咨询公司的一个招聘官，他试图根据前面的模型决定是否雇佣 George。先验知识是，招聘官相信 George 有0.3的可能性为高智商。他有 George 特定课程 Econ101 的成绩记录并看到 George 的成绩为，此时，George 有高智商的概率突然降至约。

print(infer\_student.query(['I'],evidence={'G':'C'}))

+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.9211 |  
+----------------+----------+  
| I(Intelligent) | 0.0789 |  
+----------------+----------+

注意到此时课程难度的概率也从0.4上升为0.629。

print(infer\_student.query(['D'],evidence={'G':'C'}))

+---------+----------+  
| D | phi(D) |  
+=========+==========+  
| D(Easy) | 0.3707 |  
+---------+----------+  
| D(Hard) | 0.6293 |  
+---------+----------+

现在假定招聘官丢失了 George 的成绩单，关于 George 的信息只剩下来自 Econ101 课程教授的推荐信，不过这封推荐信不是一封好的推荐信，此时，George 有高智商的概率依然会降低，为。

print(infer\_student.query(['I'],evidence={'L':'Bad'}))

+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.8600 |  
+----------------+----------+  
| I(Intelligent) | 0.1400 |  
+----------------+----------+

注意到，如果招聘官既有成绩单又有推荐信，George 有高智商的概率和只有成绩单时的概率一样，。这类从结果推理原因的查询，是**证据推理（evidential reasoning）或解释（ explanation）**。

print(infer\_student.query(['I'],evidence={'L':'Bad','G':'C'}))

+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.9211 |  
+----------------+----------+  
| I(Intelligent) | 0.0789 |  
+----------------+----------+

最后，George 向招聘官提交了他的 SAT 成绩，令人惊讶的是，他的 SAT 成绩很高。George 有高智商的概率戏剧性的从0.079直线上升至。

print(infer\_student.query(['I'],evidence={'L':'Bad','G':'C','S':'Good'}))

+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.4217 |  
+----------------+----------+  
| I(Intelligent) | 0.5783 |  
+----------------+----------+

直观上，与不好的考试成绩相比，SAT 高分更有价值的原因是低智商的学生 SAT 考试中获得高分的可能性非常低；相反，高智商的学生仍然可能获得。而聪明的学生在难度高的课程中也更有可能获得。事实上，可以看到，Econ101 是一门有难度的课程的概率已经从之前的0.629提高到了现在的。

实际上，关于成绩的变量，存在有一个因果因素，智商为另一个变量课程难度提供了信息。

print(infer\_student.query(['D'],evidence={'L':'Bad','G':'C','S':'Good'}))

+---------+----------+  
| D | phi(D) |  
+=========+==========+  
| D(Easy) | 0.2404 |  
+---------+----------+  
| D(Hard) | 0.7596 |  
+---------+----------+

一方面，；另一方面，如果发现 Econ101 是一门难度高的课程，。因此关于 George 在课程 Econ101 中的成绩，实际上给出部分解释。

print(infer\_student.query(['I'],evidence={'G':'C','D':'Hard'}))

+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.8909 |  
+----------------+----------+  
| I(Intelligent) | 0.1091 |  
+----------------+----------+

又如果，George 在课程 Econ101 中的成绩是，则。

print(infer\_student.query(['I'],evidence={'G':'B'}))

+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.8252 |  
+----------------+----------+  
| I(Intelligent) | 0.1748 |  
+----------------+----------+

另一方面，如果 Econ101 比较难，那么。实际上，已经通过课程的难度**解释消除（explained away）**了成绩低这一事件带来的影响。解释消除称为**因果间推理（ intercausal reasoning）**一般推理模式的一个示例。

print(infer\_student.query(['I'],evidence={'G':'B','D':'Hard'}))

+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.6604 |  
+----------------+----------+  
| I(Intelligent) | 0.3396 |  
+----------------+----------+

query方法返回为变量状态概率，而map\_query方法直接返回具有最大概率的状态。例如下述条件概率查询有，因此返回变量的状态为。

print(infer\_student.map\_query(['I'],evidence={'G':'B','D':'Hard','S':'Good'}))  
print(infer\_student.query(['I'],evidence={'G':'B','D':'Hard','S':'Good'}))

0it [00:00, ?it/s]  
  
  
  
0it [00:00, ?it/s]  
  
  
{'I': 'Intelligent'}  
+----------------+----------+  
| I | phi(I) |  
+================+==========+  
| I(Dumb) | 0.1084 |  
+----------------+----------+  
| I(Intelligent) | 0.8916 |  
+----------------+----------+

## 3.5.3 连续型贝叶斯网络

许多情形下，需要在某个连续空间中取值，例如位置、温度、压力及气体浓度等，显然，这些情况都无法用表的形式来表示。一种解决策略是离散所有的连续变量，不过，为了得到合理准确的模型，通常需要非常精细的离散化操作，这使得变量的离散值数量庞大；且离散化连续变量常常会丢失许多刻画其特征的结构，例如连续变量“邻近的”离散值相关联概率之间成立的某种关系等。实际上，贝叶斯网络的简洁表示并未限制随机变量只为离散型，其仅有的要求是对的每一个赋值，有表示上的一个分布，可以是连续的，但 CPD 需要在值的一个连续统（ continuum）上表示分布；可能还存在的一些连续的父节点，所以 CPD 还需要表示不同概率分布的一个连续统。

或描述为，对于随机变量的 CPD，每一赋值，有表示上的一个分布。

### 3.5.3.1 连续型随机变量的表示——ContinuousFactor方法

pgmpy库提供了ContinuousFactor方法来表征连续型随机变量的分布，输入参数variables为定义分布的变量；pdf为分布的概率密度函数。

%load\_ext autoreload   
%autoreload 2   
import usda.data\_visual as usda\_vis  
import usda.pgm as usda\_pgm  
  
import numpy as np  
from scipy.special import beta  
from scipy.stats import multivariate\_normal  
from pgmpy.factors.continuous import ContinuousFactor  
from pgmpy.factors.distributions import GaussianDistribution as JGD  
from pgmpy.factors.continuous import CanonicalDistribution  
from pgmpy.factors.continuous import LinearGaussianCPD  
from pgmpy.models import LinearGaussianBayesianNetwork  
  
import matplotlib.pyplot as plt  
import matplotlib as mpl

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

from datascience import \*  
from prob140 import \*  
%matplotlib inline  
import matplotlib.pyplot as plt  
import numpy as np  
plt.style.use('fivethirtyeight')

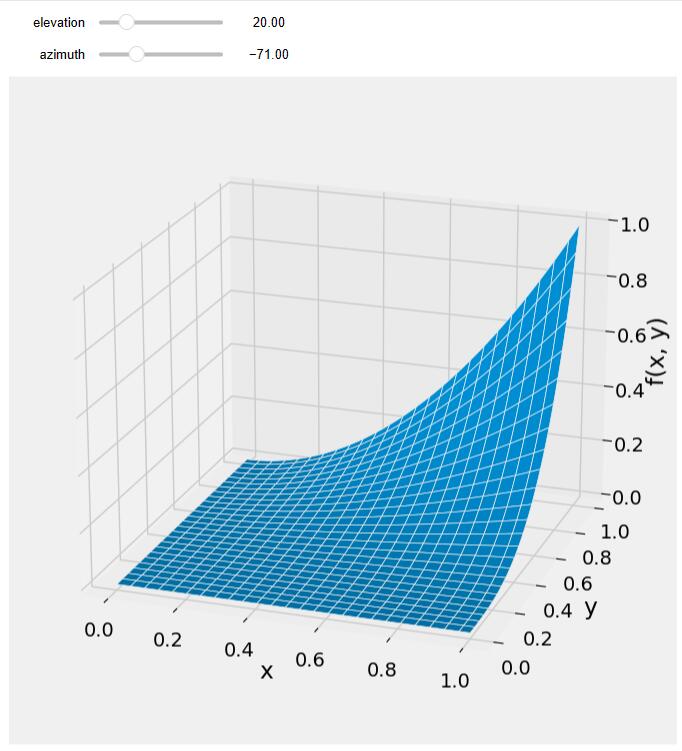
help(ContinuousFactor.\_\_init\_\_)

Help on function \_\_init\_\_ in module pgmpy.factors.continuous.ContinuousFactor:  
  
\_\_init\_\_(self, variables, pdf, \*args, \*\*kwargs)  
 Parameters  
 ----------  
 variables: list or array-like  
 The variables for which the distribution is defined.  
   
 pdf: function  
 The probability density function of the distribution.  
   
 Examples  
 --------  
 >>> import numpy as np  
 >>> from scipy.special import beta  
 >>> from pgmpy.factors.continuous import ContinuousFactor  
 # Two variable dirichlet distribution with alpha = (1,2)  
 >>> def dirichlet\_pdf(x, y):  
 ... return (np.power(x, 1) \* np.power(y, 2)) / beta(x, y)  
 >>> dirichlet\_factor = ContinuousFactor(['x', 'y'], dirichlet\_pdf)  
 >>> dirichlet\_factor.scope()  
 ['x', 'y']  
 >>> dirichlet\_factor.assignment(5,6)  
 226800.0

* 示例分布-1

定义 pdf 为函数drichlet\_pdf，包含2个变量，并打印查看分布图形。

def drichlet\_pdf(x, y):  
 return (np.power(x, 1)\*np.power(y, 2))/beta(x, y)  
   
Plot\_3d(x\_limits=(0,1), y\_limits=(0,1), f=drichlet\_pdf, cstride=4, rstride=4,interactive=True)



使用ContinuousFactor方法表征分布。可以通过assignment方式传入变量取值计算概率。

drichlet\_factor = ContinuousFactor(['x', 'y'], drichlet\_pdf)   
print(drichlet\_factor.scope(),drichlet\_factor.assignment(0.9,0.9),drichlet\_factor.assignment(0.1,0.1))

['x', 'y'] 0.594569360493 5.0723727439e-05

* 示例分布-2

使用reduce方式可以指定待消减的随机变量一个值后，消减该变量。例如对含有3个变量的分布指定0.8值消减后，转变为仅含有的2个变量的分布。

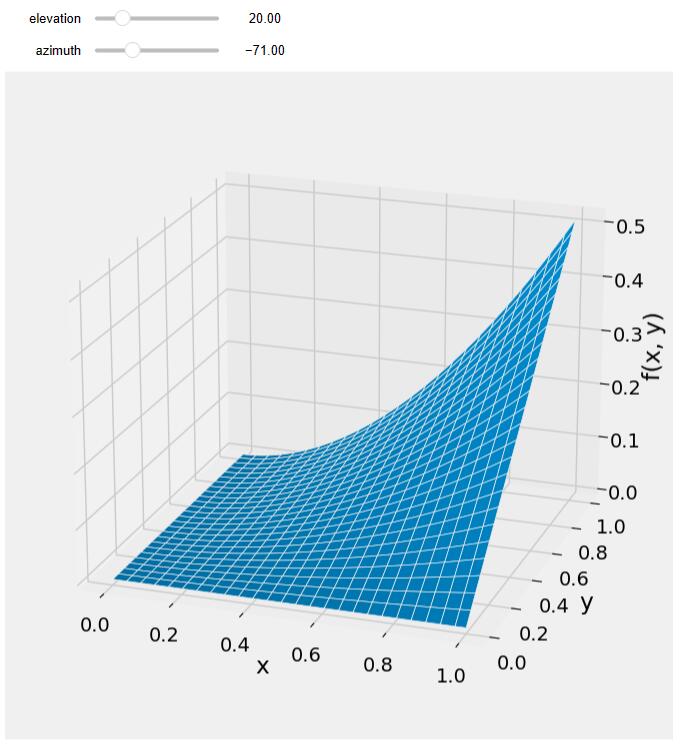
def custom\_pdf(x, y, z):  
 return z\*(np.power(x, 1)\*np.power(y, 2))/beta(x, y)  
  
custom\_factor = ContinuousFactor(['x', 'y', 'z'], custom\_pdf)  
print(custom\_factor.scope(),custom\_factor.assignment(0.8, 0.8, 0.8))

['x', 'y', 'z'] 0.270012958213

custom\_factor.reduce([('y', 0.8)])  
print(custom\_factor.scope(),custom\_factor.assignment(0.8,0.8))

['x', 'z'] 0.270012958213

Plot\_3d(x\_limits=(0,1), y\_limits=(0,1), f=custom\_factor.pdf, cstride=4, rstride=4,interactive=True)



* 示例分布-3

使用SciPy库的multivariate\_normal方法建立多个随机变量的多元正态分布，输入参数mean为分布的均值；cov为对称正定（半正定）协方差矩阵（Symmetric positive (semi)definite covariance matrix）。

关于*正定（半正定）协方差矩阵*参考本章最后部分。

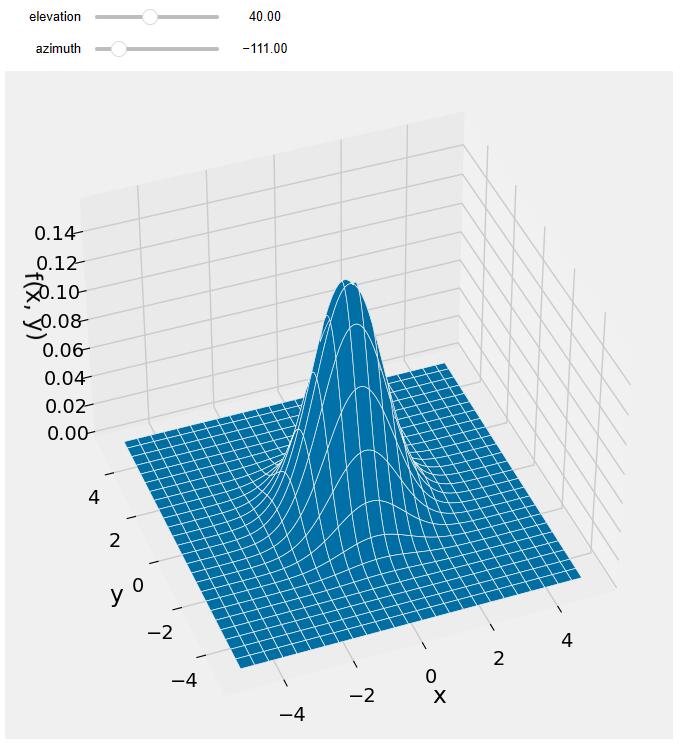
help(multivariate\_normal.pdf)

Help on method pdf in module scipy.stats.\_multivariate:  
  
pdf(x, mean=None, cov=1, allow\_singular=False) method of scipy.stats.\_multivariate.multivariate\_normal\_gen instance  
 Multivariate normal probability density function.  
   
 Parameters  
 ----------  
 x : array\_like  
 Quantiles, with the last axis of `x` denoting the components.  
 mean : array\_like, default: ``[0]``  
 Mean of the distribution.  
 cov : array\_like, default: ``[1]``  
 Symmetric positive (semi)definite covariance matrix of the distribution.  
 allow\_singular : bool, default: ``False``  
 Whether to allow a singular covariance matrix.  
   
 Returns  
 -------  
 pdf : ndarray or scalar  
 Probability density function evaluated at `x`  
   
 Notes  
 -----  
 Setting the parameter `mean` to `None` is equivalent to having `mean`  
 be the zero-vector. The parameter `cov` can be a scalar, in which case  
 the covariance matrix is the identity times that value, a vector of  
 diagonal entries for the covariance matrix, or a two-dimensional  
 array\_like.

std\_normal\_pdf = lambda \*x: multivariate\_normal.pdf(x, [0, 0], [[1, 0], [0, 1]])  
std\_normal = ContinuousFactor(['x1', 'x2'], std\_normal\_pdf)  
print(std\_normal.scope(),std\_normal.assignment([0.5, 0.5]))

['x1', 'x2'] 0.12394999431

Plot\_3d(x\_limits=(-5,5), y\_limits=(-5,5), f=std\_normal.pdf, cstride=4, rstride=4,interactive=True)

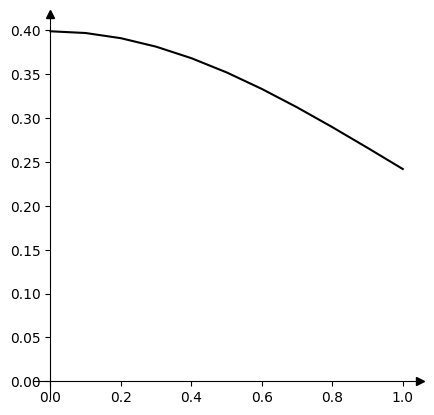


计算服从正态分布的二维随机变量中关于的边缘概率分布。

std\_normal.marginalize(['x2'])  
print(std\_normal.scope(), std\_normal.assignment(0.5))

['x1'] 0.35206532676429947

mpl.rcParams.update(mpl.rcParamsDefault)  
fig, ax=plt.subplots(1, 1,figsize=(5,5))  
x=np.arange(0,1+0.1,0.1)  
  
ax.plot(x, list(map(std\_normal.pdf,x)), 'k-', ms=8)  
usda\_vis.plot\_style\_axis\_A(ax)  
plt.show()



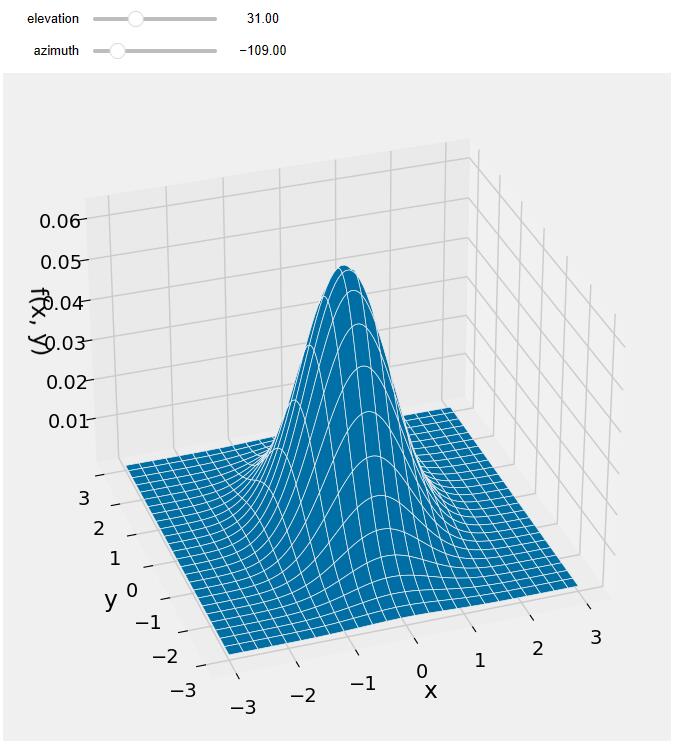
* 示例分布-4

ContinuousFactor类含有对分布执行乘除等计算。

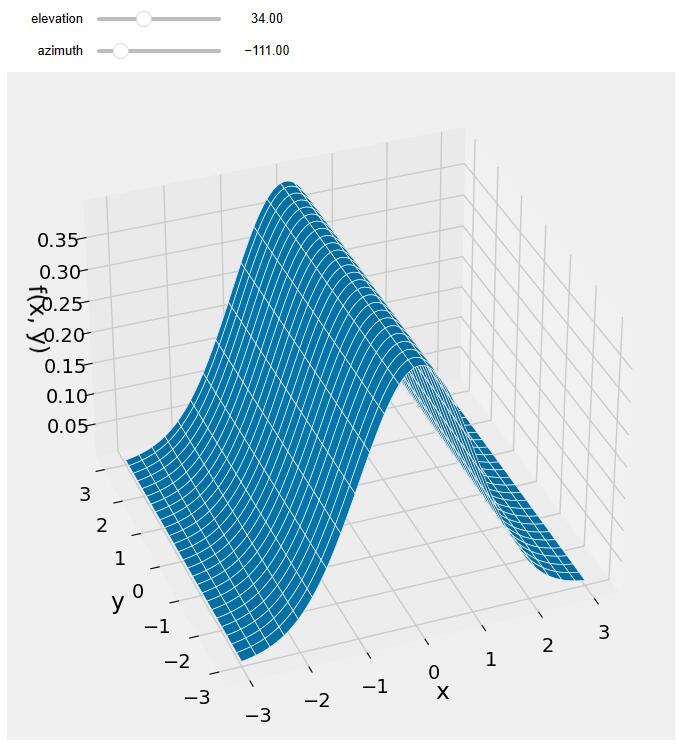
sn\_pdf1 = lambda x: multivariate\_normal.pdf([x], [0], [[1]])  
sn\_pdf2 = lambda x1,x2: multivariate\_normal.pdf([x1, x2], [0, 0], [[1, 0], [0, 1]])  
sn1 = ContinuousFactor(['x2'], sn\_pdf1)  
sn2 = ContinuousFactor(['x1', 'x2'], sn\_pdf2)  
sn3 = sn1 \* sn2  
sn4 = sn2 / sn1  
print(sn3.scope(),sn3.assignment(0.6, 0.6))  
print(sn4.scope(), sn4.assignment(0.6, 0.6))

['x2', 'x1'] 0.0370008053775  
['x1', 'x2'] 0.333224602892

Plot\_3d(x\_limits=(-3,3), y\_limits=(-3,3), f=sn3.pdf, cstride=4, rstride=4,interactive=True)



Plot\_3d(x\_limits=(-3,3), y\_limits=(-3,3), f=sn4.pdf, cstride=4, rstride=4,interactive=True)



### 3.5.3.2 多元高斯分布和CPD

#### 1）多元高斯分布

连续型变量的参数化空间在本质上是无界的，其中多元高斯分布类是简单的一种分布子类，具有很强的假设，例如分布偏离其均值的指数式衰减，及变量之间线性的交互作用等。在*概率论基础*部分阐述了一维和二维正态（高斯）分布，前文的示例分布3和4也使用SciPy库提供的方法建立二元和多元高斯分布。根据均值和方差这两个参数定义单变量高斯分布，对多个随机变量的多元高斯分布，则由维的均值向量（ mean vector），及对称协方差矩阵（covariance matrix）定义，密度函数定义如下：，式中是的行列式。

把标准高斯分布的概念推广到多维情况，将其定义为一个均值向量为0，协方差矩阵为对角线元素是1，其余元素是0的单位矩阵的高斯分布。多维标准高斯分布仅是每一维上独立标准高斯分布的乘积。为了使这个等式导出定义明确积分为1的密度函数，矩阵必须是正定的：对于任意满足的（列向量），有。由于正定矩阵是非奇异的，因此其行列式大于0，这与该定义的必要条件一致。将多元高斯分布推广为半正定协方差矩阵的情况：对于任意的，有，其允许一些应用中的协方差矩阵是奇异的。

例如考虑三个随机变量上的一个特殊联合分布，均值向量和协方差矩阵为：

。使用pgmpy库提供的GaussianDistribution（JGD）方法实现。

cov=np.array([[4, 2, -2],   
 [2, 5, -5],   
 [-2, -5, 8]])  
dis = JGD(['x1', 'x2', 'x3'], np.array([[1], [-3], [4]]),cov)  
print(dis.variables)  
print(dis.mean)  
print(dis.covariance)  
print(dis.pdf([0.5,0.5,0.5]))

['x1', 'x2', 'x3']  
[[ 1.]  
 [-3.]  
 [ 4.]]  
[[ 4. 2. -2.]  
 [ 2. 5. -5.]  
 [-2. -5. 8.]]  
0.00153154445492

检查协方差矩阵是否是正定/半正定矩阵。

usda\_pgm.covariance\_test(cov,[0,1])

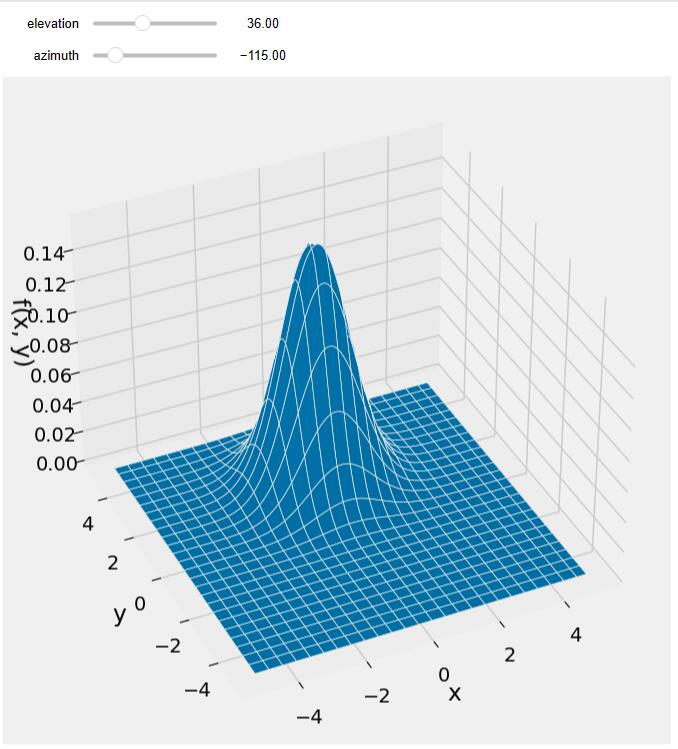
{'Positive Definite': True,  
 'Positive Semi-Definite': True,  
 'Symmetric': True,  
 'Positive Determinant': True,  
 'Eigen Values, Positivity': True}

同ContinuousFactor类，由GaussianDistribution定义的高斯分布同样也含有边缘（高斯）分布（marginalize）、消减（reduce）、标准化（normalize）、乘除（product and divide）等运算。

dis1 = JGD(['x1', 'x2', 'x3'], np.array([[1], [-3], [4]]),np.array([[4, 2, -2], [2, 5, -5], [-2, -5, 8]]))  
dis2 = JGD(['x3', 'x4'], [0, 2], [[1, 0], [0, 1]])  
dis3 = dis1 \* dis2  
print(dis3.variables)  
print(dis3.mean)  
print(dis3.covariance)

['x1', 'x2', 'x3', 'x4']  
[[ 1.88888889]  
 [-0.77777778]  
 [ 0.44444444]  
 [ 2. ]]  
[[ 3.55555556 0.88888889 -0.22222222 0. ]  
 [ 0.88888889 2.22222222 -0.55555556 0. ]  
 [-0.22222222 -0.55555556 0.88888889 0. ]  
 [ 0. 0. 0. 1. ]]

dis2.normalize(inplace=True)  
Plot\_3d(x\_limits=(-5,5), y\_limits=(-5,5), f=dis2.pdf, cstride=4, rstride=4,interactive=True)



#### 2）高斯密度的信息形式和CanonicalDistribution

由于正定矩阵是可逆的，如果高斯分布根据协方差矩阵的逆矩阵定义，称为**信息矩阵**（information matrix）或**精度矩阵**（ precision matrix）。这种表示可以为高斯密度导出另一种形式，上述密度函数的指数有：，式中最后一项是常数，所以，。该高斯密度公式通常称为**信息形式**（information form），并且向量称为**位势向量**（ potential vector）。因为正定当且仅当正定，所以信息形式定义了一个有效的高斯密度，当且仅当信息矩阵对称且正定。

根据高斯密度的信息形式，pgmpy提供了CanonicalDistribution方法以信息形式表述高斯密度分布。其输入参数K即J，为协方差矩阵的逆矩阵；h即位势向量；g为常数。

help(CanonicalDistribution.\_\_init\_\_)

Help on function \_\_init\_\_ in module pgmpy.factors.distributions.CanonicalDistribution:  
  
\_\_init\_\_(self, variables, K, h, g)  
 Parameters  
 ----------  
 variables: list or array-like  
 The variables for wich the distribution is defined.  
   
 K: n x n, 2-d array-like  
   
 h : n x 1, array-like  
   
 g : int, float  
   
 pdf: function  
 The probability density function of the distribution.  
   
 The terms K, h and g are defined parameters for canonical  
 factors representation.  
   
 Examples  
 --------  
 >>> from pgmpy.factors.continuous import CanonicalDistribution  
 >>> phi = CanonicalDistribution(['X', 'Y'], np.array([[1, -1], [-1, 1]]),  
 np.array([[1], [-1]]), -3)  
 >>> phi.variables  
 ['X', 'Y']  
   
 >>> phi.K  
 array([[1, -1],  
 [-1, 1]])  
   
 >>> phi.h  
 array([[1],  
 [-1]])  
   
 >>> phi.g  
 -3

phi1 = CanonicalDistribution(['x1', 'x2', 'x3'],np.array([[1, -1, 0], [-1, 4, -2], [0, -2, 4]]),np.array([[1], [4], [-1]]), -2)  
phi2 = CanonicalDistribution(['x1', 'x2'], np.array([[3, -2], [-2, 4]]),np.array([[5], [-1]]), 1)  
  
phi3 = phi1 \* phi2  
print(phi3.variables,'\n---',phi3.h,'\n---',phi3.K,'\n---',phi3.g)

['x1', 'x2', 'x3']   
--- [[ 6.]  
 [ 3.]  
 [-1.]]   
--- [[ 4. -3. 0.]  
 [-3. 8. -2.]  
 [ 0. -2. 4.]]   
--- -1

用to\_joint\_gaussian方法可以将高斯密度的信息形式转换回一般定义形式。

phi = CanonicalDistribution(['x1', 'x2'], np.array([[3, -2], [-2, 4]]),np.array([[5], [-1]]), 1)  
jgd = phi.to\_joint\_gaussian()  
print(jgd.variables)  
print(jgd.covariance)  
print(jgd.mean)

['x1', 'x2']  
[[ 0.5 0.25 ]  
 [ 0.25 0.375]]  
[[ 2.25 ]  
 [ 0.875]]

#### 3）二元高斯分布参数的矩阵表示

二元高斯分布的均值向量和协方差矩阵分别对应正态分布的一阶矩和二阶矩，用矩阵的符号表示有，，。这种表示在单变量的层次上分解，有是的均值，是的方差，且是和之间的协方差：。

#### 4）高斯分布的独立性

对于多元高斯分布，独立性可以根据分布的参数直接确定。

定理：另X=X1​,⋯,Xn​ (; )XiXj 。

用信息矩阵方式表述。

定理：考虑高斯分布，且令为其信息矩阵，则，当且仅当。

#### 5）线性高斯模型和CPD

考虑连续变量关于连续型父节点的依赖，可以把对的分布构建为其参数取决于值的高斯模型，一般的方式是决定的均值是的线性函数，且的方差不依赖于，例如。这类独立性称为线性高斯（Linear Gaussian）模型。

定义：另有连续型父节点的连续型变量，假如存在参数β0​,…,βkY。

用LinearGaussianCPD方法定义线性高斯模型（CPD）。

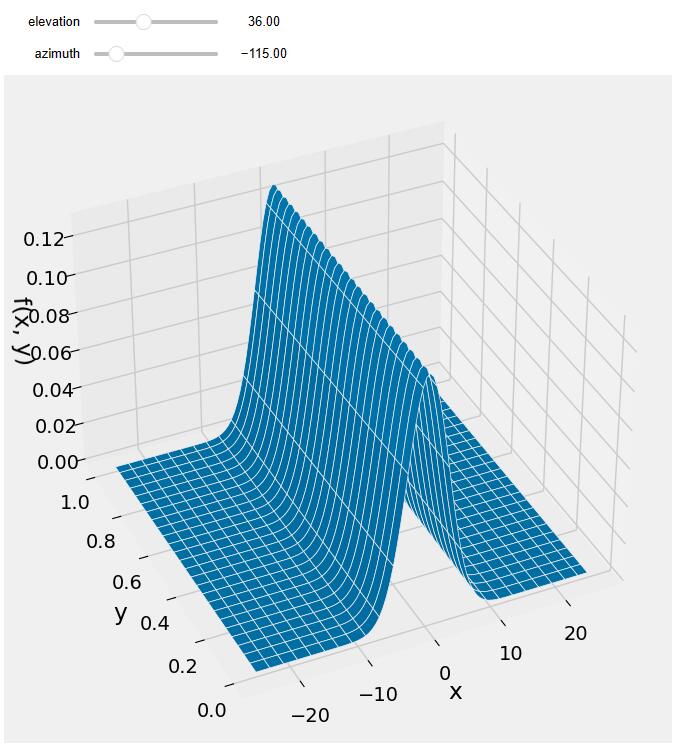
help(LinearGaussianCPD.\_\_init\_\_)

Help on function \_\_init\_\_ in module pgmpy.factors.continuous.LinearGaussianCPD:  
  
\_\_init\_\_(self, variable, evidence\_mean, evidence\_variance, evidence=[], beta=None)  
 Parameters  
 ----------  
   
 variable: any hashable python object  
 The variable whose CPD is defined.  
   
 evidence\_mean: Mean vector (numpy array) of the joint distribution, X  
   
 evidence\_variance: int, float  
 The variance of the multivariate gaussian, X = ['x1', 'x2', ..., 'xn']  
   
 evidence: iterable of any hashable python objects  
 An iterable of the parents of the variable. None if there are no parents.  
   
 beta (optional): iterable of int or float  
 An iterable representing the coefficient vector of the linear equation.  
 The first term represents the constant term in the linear equation.  
   
   
 Examples  
 --------  
   
 # For P(Y| X1, X2, X3) = N(-2x1 + 3x2 + 7x3 + 0.2; 9.6)  
   
 >>> cpd = LinearGaussianCPD('Y', [0.2, -2, 3, 7], 9.6, ['X1', 'X2', 'X3'])  
 >>> cpd.variable  
 'Y'  
 >>> cpd.evidence  
 ['x1', 'x2', 'x3']  
 >>> cpd.beta\_vector  
 [0.2, -2, 3, 7]

cpd = LinearGaussianCPD('Y', [0.2, -2, 3, 7], 9.6, ['X1', 'X2', 'X3'])  
print(cpd)

P(Y | X1, X2, X3) = N(-2\*X1 + 3\*X2 + 7\*X3 + 0.2; 9.6)

Plot\_3d(x\_limits=(-25,25), y\_limits=(0,1), f=cpd.pdf, cstride=4, rstride=4,interactive=True)



### 3.5.3.3 高斯贝叶斯网

基于线性高斯模型用贝叶斯网定义连续的联合分布。

定义：**高斯贝叶斯网**定义为其所有变量都是连续的，且其中的所有 CPD 都是线性高斯的贝叶斯网。

定理：令是其父节点的线性高斯，假设是具有分布N(μ;Σ)的联合高斯，则：

* 的分布是一个正态分布，式中，；
* 上的联合分布是一个正态分布，其中。

例如，对于线性高斯网，有p(X2​∣X1​)=N(0.5X1​−3.5;4) P(X3​∣X2​)=N(−X2​+1;3)，根据上述定理，可以计算高斯分布。对于均值有，μ2​=0.5μ1​−3.5=0.5⋅1−3.5=−3μ3​=(−1)μ2​+1=(−1)⋅(−3)+1=4

和的方差计算为：Σ22​=4+(1/2)2⋅4=5Σ33​=3+(−1)2⋅5=8，可以看到变量的方差是两项和：源自自身高斯噪声（Gaussian noise）参数的方差和源自由依赖性强度加权其父节点变量的方差。协方差的计算结果如下：Σ12​=(1/2)⋅4=2 Σ23​=(−1)⋅Σ22​=−5 Σ13​=(−1)⋅Σ12​=−2

上述计算过程，可以由pgmpy库的LinearGaussianBayesianNetwork方法建立高斯贝叶斯网（高斯 BN），通过to\_joint\_gaussian方法得到联合高斯分布P(X1, X2, X3)后计算。

model = LinearGaussianBayesianNetwork([('x1', 'x2'), ('x2', 'x3')])  
cpd1 = LinearGaussianCPD('x1', [1], 4)  
cpd2 = LinearGaussianCPD('x2', [-3.5, 0.5], 4, ['x1'])  
cpd3 = LinearGaussianCPD('x3', [1, -1], 3, ['x2'])  
  
cpd1.variables = [\*cpd1.evidence, cpd1.variable]  
cpd2.variables = [\*cpd2.evidence, cpd2.variable]  
cpd3.variables = [\*cpd3.evidence, cpd3.variable]  
model.add\_cpds(cpd1, cpd2, cpd3)  
jgd = model.to\_joint\_gaussian()  
print(jgd.variables)  
print(jgd.mean)  
print(jgd.covariance)  
  
for i in model.cpds:  
 print(i)

['x1', 'x2', 'x3']  
[[ 1.]  
 [-3.]  
 [ 4.]]  
[[ 4. 2. -2.]  
 [ 2. 5. -5.]  
 [-2. -5. 8.]]  
P(x1) = N(1; 4)  
P(x2 | x1) = N(0.5\*x1 + -3.5; 4)  
P(x3 | x2) = N(-1\*x2 + 1; 3)

## 3.5.4 学习（Learning）

前文叙述的贝叶斯网络（BN）直接构建（含结构（图）和参数（CPD））后再进行推理分析的方式，实际上并不实用。原因是参数的配置（CPDs）由人工完成并不现实，一方面由各个领域专家参与相关模型的参数评估不容易实现，也会耗费过多的时间成本，且配置的参数需要通过不断的测试验证，其准确性不容易达到；另外，可能由于情况的变化，模型往往需要重新设计，这也为直接建立结构和参数的模型带来困难。然而由于数据的不断积累，从数据中学习模型的结构和参数成为可能，此类建立模型的方法称为**模型学习（model learning）**。

模型学习涉及两个方面：

1. 参数估计（Parameter Estimation）：给定一组数据样本和一个可以捕获变量之间依赖关系的 DAG，估计各个变量的（条件）概率分布（CPDs）。一般包括最大似然估计（Maximum Likelihood Estimation，MLE）和贝叶斯参数估计（Bayesian Parameter Estimation）等方法；
2. 结构学习（Structure Learning）：给定一组数据样本，估计一个可以捕获变量之间依赖关系的 DAG 及变量之间的（条件）概率分布。一般包括基于得分（Score-based）的结构估计，基于约束（Constrain-based）的结构估计（PC）和混合（Hybrid）结构估计（MMHC）。

### 3.5.4.1 参数估计[5,2,1]

数据集，为随机变量（random variable, r.v.）的一系列独立同分布（independent and identically distribute，i.i.d）的实现。参数依赖于所考虑的一个分布，例如，高斯分布等。对于这些数据（）和参数（），根据贝叶斯公式：，式中，和为随机事件，且：

* 为条件概率（ conditional probability），在事件发生条件下事件发生的概率，也称为给定事件下事件的后验概率（posterior probability）；
* 同为条件概率，在事件发生条件下事件发生的概率，因为，也被解释为固定给定的，对于的似然（likelihood）；
* 和是不给定任何条件下，观测到的事件和各自的概率，也称为先验概率（prior probability）和边缘概率（marginal probability）。

有，因此用术语表述有：。

下述试验中，提供了两个案例，分布对应两组数据，一组来自于 Heinrich, G 论文不均匀硬币抛掷试验，coin\_data，仅包含一个随机变量，含两个状态，正面head用1表示，反面tail用0表示；另一组来自于pgmpy库说明文件中水果味道的数据，fruit\_data，包含有3个随机变量，每个随机变量各包含两个状态，分别为banana,apple，large,small和yes,no。

%load\_ext autoreload   
%autoreload 2   
import usda.data\_visual as usda\_vis  
import usda.pgm as usda\_pgm  
  
import daft  
from pgmpy.models import BayesianNetwork  
from pgmpy.factors.discrete import TabularCPD  
from pgmpy.inference import VariableElimination,BeliefPropagation  
from pgmpy.estimators import ParameterEstimator,MaximumLikelihoodEstimator,BayesianEstimator  
  
import pandas as pd  
from scipy.stats import bernoulli,beta,binom  
import matplotlib.pyplot as plt  
import numpy as np  
from IPython.display import display  
from sklearn import preprocessing  
  
import sympy  
from sympy import Symbol, simplify, pprint, factor,solve,log  
from sympy.stats import Beta, density, E, variance

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

coin\_data=pd.DataFrame(bernoulli.rvs(0.5, size=20),columns=['tossing'])  
print(coin\_data)  
  
print('-'\*50)  
fruit\_data = pd.DataFrame(data={'fruit': ["banana", "apple", "banana", "apple", "banana","apple", "banana",   
 "apple", "apple", "apple", "banana", "banana", "apple", "banana",],   
 'tasty': ["yes", "no", "yes", "yes", "yes", "yes", "yes",   
 "yes", "yes", "yes", "yes", "no", "no", "no"],   
 'size': ["large", "large", "large", "small", "large", "large", "large",  
 "small", "large", "large", "large", "large", "small", "small"]})  
print(fruit\_data)

tossing  
0 1  
1 1  
2 0  
3 0  
4 1  
5 0  
6 0  
7 1  
8 1  
9 1  
10 1  
11 0  
12 1  
13 1  
14 1  
15 0  
16 0  
17 1  
18 0  
19 1  
--------------------------------------------------  
 fruit tasty size  
0 banana yes large  
1 apple no large  
2 banana yes large  
3 apple yes small  
4 banana yes large  
5 apple yes large  
6 banana yes large  
7 apple yes small  
8 apple yes large  
9 apple yes large  
10 banana yes large  
11 banana no large  
12 apple no small  
13 banana no small

#### 1）最大似然估计（MLE）

MLE 试图找到使得似然最大化的参数，有，即从随机变量中观测到的事件数据集的联合概率。因为似然函数和似然函数的对数最大化（达到最大值）所对应的参数相同（可以查看*2.6.2 逻辑回归二分类到 SoftMax 回归多分类*部分的图表），且与原始似然函数相比，似然函数的对数求导更容易，因此，MLE 可写为：。对参数估计就是对似然函数求导，有。

根据已知数据集估计参数后，可以预测一个新观测值的概率，有p(x~∣X)=∫ϑ∈Θ​p(x~∣ϑ)p(ϑ∣X)dϑ≈∫ϑ∈Θ​p(x~∣ϑ^ML​)p(ϑ∣X)dϑ=p(x~∣ϑ^ML​)。

不均匀硬币抛掷试验（伯努利试验），估计未知参数。由伯努利分布（伯努利密度函数），对于随机变量有，，式中，定义为正面head，为背面tail。对参数构建 MLE，为数据的一个函数，有n(c) cp。

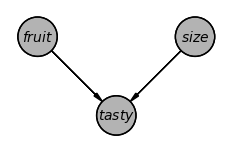
例如，对于数据集coin\_data，通过直接计算频率，及应用pgmpy库分别计算，计算结果同，正面朝上的概率（参数）为0.598。

print(coin\_data.value\_counts(normalize =True))  
  
coin\_bn=BayesianNetwork()  
coin\_bn.add\_node('tossing')  
coin\_bn.fit(coin\_data, estimator=MaximumLikelihoodEstimator) # `BayesianNetwork`类内置的`fit`方法提供了参数估计方便的操作模式。  
print(coin\_bn.cpds[0])

tossing  
1 0.6  
0 0.4  
dtype: float64  
+------------+-----+  
| tossing(0) | 0.4 |  
+------------+-----+  
| tossing(1) | 0.6 |  
+------------+-----+

对于水果味道观测试验，定义贝叶斯网络（BN），含有3个随机变量（节点），有边，为v-结构。且已知数据data，估计参数，即 CPDs。为了理解给定的数据，根据父节点计算子节点对应的数量，可知共有apple7个，banana7个，总共14个；其中对于状态apple，当状态为large时，对于状态为yes有3个，no的有1个，其它情况以此类推。

fruit\_bn = BayesianNetwork([('fruit', 'tasty'), ('size', 'tasty')])  
fruit\_bn\_daft=fruit\_bn.to\_daft(node\_pos={'fruit':(1,2),'size':(3,2),'tasty':(2,1)})  
fruit\_bn\_daft.render(dpi=100);



fruit\_bn\_pe=ParameterEstimator(fruit\_bn, fruit\_data)  
print("\n", fruit\_bn\_pe.state\_counts('fruit')) # unconditional  
print("\n", fruit\_bn\_pe.state\_counts('tasty')) # conditional on fruit and size

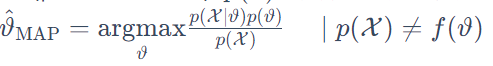
fruit  
apple 7  
banana 7  
  
 fruit apple banana   
size large small large small  
tasty   
no 1.0 1.0 1.0 1.0  
yes 3.0 2.0 5.0 0.0

用MaximumLikelihoodEstimator方法根据定义的图结构和反映随机变量间关系的数据fruit\_data，直接计算 BN 的 CPDs 如下。

fruit\_bn\_fitted = MaximumLikelihoodEstimator(fruit\_bn, fruit\_data)  
print(fruit\_bn\_fitted.estimate\_cpd('fruit'))   
print('/'\*50)  
for cpd in fruit\_bn\_fitted.get\_parameters():  
 print(cpd)

+---------------+-----+  
| fruit(apple) | 0.5 |  
+---------------+-----+  
| fruit(banana) | 0.5 |  
+---------------+-----+  
//////////////////////////////////////////////////  
+---------------+-----+  
| fruit(apple) | 0.5 |  
+---------------+-----+  
| fruit(banana) | 0.5 |  
+---------------+-----+  
+------------+--------------+--------------------+---------------------+---------------+  
| fruit | fruit(apple) | fruit(apple) | fruit(banana) | fruit(banana) |  
+------------+--------------+--------------------+---------------------+---------------+  
| size | size(large) | size(small) | size(large) | size(small) |  
+------------+--------------+--------------------+---------------------+---------------+  
| tasty(no) | 0.25 | 0.3333333333333333 | 0.16666666666666666 | 1.0 |  
+------------+--------------+--------------------+---------------------+---------------+  
| tasty(yes) | 0.75 | 0.6666666666666666 | 0.8333333333333334 | 0.0 |  
+------------+--------------+--------------------+---------------------+---------------+  
+-------------+----------+  
| size(large) | 0.714286 |  
+-------------+----------+  
| size(small) | 0.285714 |  
+-------------+----------+

#### 2）最大后验估计（Maximum a posteriori estimation，MAP(E)）

MAP 类似于 MLE，但允许通过使用先验分布（prior distribution）对参数进行加权，为给定数据条件下最大化参数后验的目标，。通过贝叶斯公式，有。

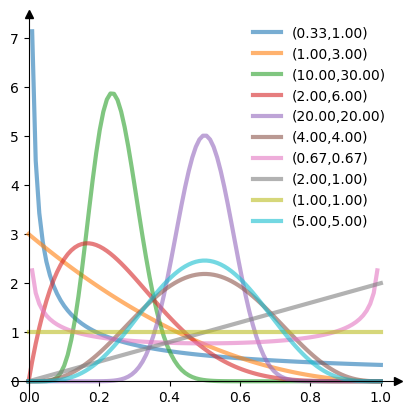
一个先验分布被加到似然项上，通过这额外的先验知识平衡数据集的观测结果，避免 MLE 这一简单模型的过拟合现象。对的数据建模延续贝叶斯方法，将看作随机变量，有  
p(ϑ):=p(ϑ∣α)αp(ϑ)  
ϑ预期值的一个信念（belief）。

MAP 通过最大化估计参数。参数估计后，同 MLE，可以计算给定观测值的概率，，式中，为参数空间。

返回硬币抛掷试验。因为我们根据经验相信在抛掷硬币时，有0.5的几率为正面，0.5的几率为背面，选择 Beta 分布定义的先验分布有，，式中，。对于 Beta 的概率密度函数（PDF）可书写为：，式中，为伽马函数（gamma function ），定义为，。

使用SciPy库，给定不同的值，打印观察 PDF 曲线。因为相信一半的几率正面朝上，因此配置超参数（对应），其 PDF 曲线最大值位于0.5处，即预先相信正面朝上为0.5概率的可能性最大。最大值为0.5的 Beta 密度函数并不唯一，只要满足值相同，其差异是曲线的强度（峰度）变化。一般而言，对于少量样本数据使用峰度偏小的分布曲线，而大量样本数据使用峰度偏大的分布曲线。

fig, ax=plt.subplots(1, 1,figsize=(5,5))  
  
ab\_lst=[(1/3,1),(1,3),(10,30),(2,6),(20,20),(4,4),(2/3,2/3),(2,1),(1,1),(5,5)]  
x = np.linspace(0, 1, 100)  
  
for a,b in ab\_lst:  
 ax.plot(x, beta.pdf(x, a, b), lw=3, alpha=0.6, label=f'({a:.2f},{b:.2f})')  
  
usda\_vis.plot\_style\_axis\_A(ax)  
plt.legend(loc='upper right',frameon=False)  
plt.show()



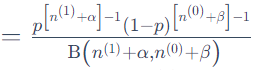
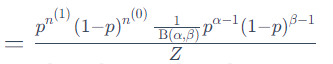
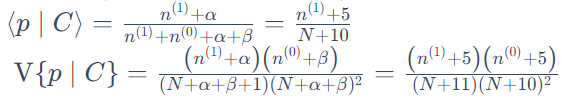
推导增加所估计参数的先验分布后，找到 MAP 最大化的参数为，。依据该公式，MAP 参数估计的结果为0.571，相对于 MLE 的0.6结果，估计的参数受到先验信念（prior belief）的影响，趋向于分布最大值对应的参数值（该试验中为0.5）。

head\_num,tail\_num=coin\_data.value\_counts()  
print(head\_num,tail\_num)  
tossing\_map\_theta=(head\_num+4)/(head\_num+tail\_num+8)  
print(tossing\_map\_theta)

12 8  
0.5714285714285714

#### 3）贝叶斯参数估计（Bayesian Parameter Estimation，BPE）

贝叶斯参数估计（BPE）通过允许参数集服从一定的的概率分布，而不是直接进行参数值估计来拓展 MAP 的方法。MLE 和 MAP 都求出了参数的值，BPE 则根据参数的先验分布和观察到的随机变量数据集，求取参数的后验分布，然后求出的期望值作为最终值。同时定义了一个参数的方差量来评估参数估计的质量或置信度（conﬁdence）。依据贝叶斯公式计算后验分布有，。因为并不求后验分布的最大化值，因此需要计算式中的项，由全概率公式展开可得。当观测到新的数据，后验概率自动随之调整。用 BPE 预测新观测到的数据的概率，有，式中，后验概率替换了参数值的显示计算（explicit calculation）。通过对积分，先验信念（prior belief）自动的合并到预测中，而预测自身是的一个分布。

继续硬币抛掷的试验，有。对Beta(α,β)分布有均值，  
⟨p∣α,β⟩=α(α+β)−1；方差   
，因此有。

下述代码试验，包含使用pgmpy库提供的BayesianEstimator方法和fit参数estimator=BayesianEstimator传入的方式。同时在图表打印时，给出了部分公式计算的方式。

coin\_be=BayesianEstimator(coin\_bn, coin\_data)  
coin\_be\_p=coin\_be.estimate\_cpd('tossing', prior\_type='BDeu', equivalent\_sample\_size=10)  
print(coin\_be\_p)

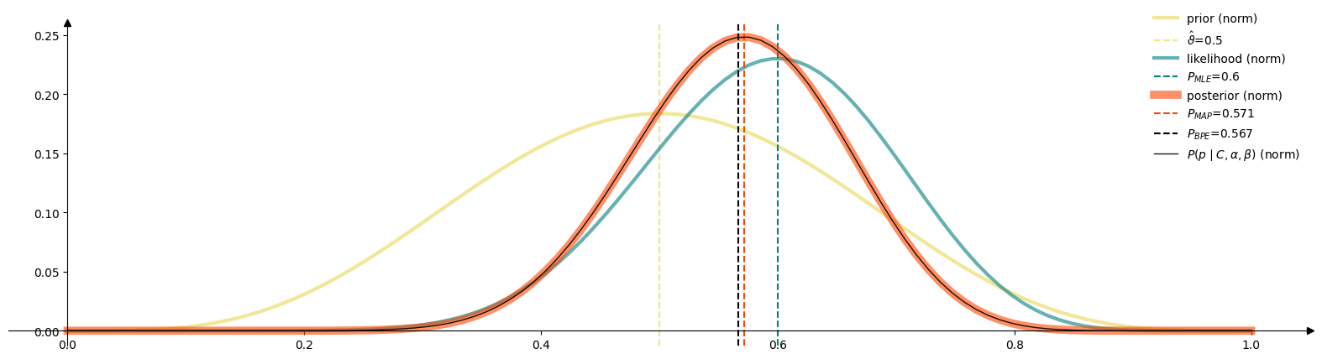
+------------+----------+  
| tossing(0) | 0.433333 |  
+------------+----------+  
| tossing(1) | 0.566667 |  
+------------+----------+

fig, ax=plt.subplots(1, 1,figsize=(5\*4,5))  
x = np.linspace(0, 1, 100)  
  
alpha\_, beta\_=sympy.symbols('alpha beta', positive=True)  
x\_, p\_=sympy.symbols('x p', positive=True)  
  
# prior  
beta\_dis=sympy.Piecewise(  
 (0, p\_ < 0),  
 (0, p\_ > 1),  
 (p\_\*\*(alpha\_-1)\*(1-p\_)\*\*(beta\_-1)/(sympy.gamma(alpha\_)\*sympy.gamma(beta\_)/sympy.gamma(alpha\_+beta\_)), True))  
print('prior density:')  
display(beta\_dis)  
beta\_dis\_=sympy.lambdify([p\_,alpha\_,beta\_],beta\_dis,"numpy")  
a,b=5,5  
ax.plot(x,preprocessing.normalize([beta\_dis\_(x, a, b)])[0], lw=3, alpha=0.9, label=f'prior (norm)',color='khaki')   
ax.axvline(x=0.5,color='khaki', linestyle="--",label='$\\hat{\\vartheta}$=0.5')  
  
# likelihood (normalized)  
pmf=p\_\*\*x\_\*(1-p\_)\*\*(1-x\_)  
L=np.prod([pmf.subs(x\_, i) for i in coin\_data['tossing']])  
L\_=sympy.lambdify(p\_,L,"numpy")  
print('Likelihood density:')  
display(L)  
ax.plot(x,preprocessing.normalize([L\_(x)])[0], lw=3, alpha=0.6, label=f'likelihood (norm)',color='teal')   
ax.axvline(x=coin\_bn.get\_state\_probability({'tossing':1}),color='teal', linestyle="--",label='$P\_{{MLE}}$={}'.format(coin\_bn.get\_state\_probability({'tossing':1})))  
  
# posterior  
posterior=L\*beta\_dis   
print('posterior density:')  
display(posterior)  
posterior\_=sympy.lambdify([p\_,alpha\_,beta\_],posterior,"numpy")  
ax.plot(x,preprocessing.normalize([posterior\_(x,a,b)])[0], lw=7, alpha=0.6, label=f'posterior (norm)',color='orangered')  
ax.axvline(x=tossing\_map\_theta,color='orangered', linestyle="--",label='$P\_{{MAP}}$={:.3f}'.format(tossing\_map\_theta))  
  
# Bayesian Parameter Estimation  
ax.axvline(x=coin\_be\_p.values[1],color='k', linestyle="--",label='$P\_{{BPE}}$={:.3f}'.format(coin\_be\_p.values[1]))  
ax.plot(x,preprocessing.normalize([beta\_dis\_(x, a+head\_num, b+tail\_num)])[0], lw=1, alpha=0.9, label=f'$P(p \\mid C, \\alpha, \\beta)$ (norm)',color='k')   
  
usda\_vis.plot\_style\_axis\_A(ax)  
plt.legend(loc='right',frameon=False,bbox\_to\_anchor=(1, 0.8))  
plt.show()

prior density:

Likelihood density:

posterior density:



对水果味道的试验，从上述 MLE 的表现来看，存在对数据过度拟合的问题。例如其中对于的状态banana，当的状态为small时，状态yes的概率为0。这里，只观测到了香蕉为小个时的1个样本为好吃，因为样本量少，这很难确定小个香蕉是否不好吃。如果观测到的数据不能代表变量的潜在分布，则 MLE 估计的结果往往并不理想。

在估计 BN 参数时，缺乏数据往往是常见的一个问题；另，如果一个变量节点有3个父节点，每个父节点各含有10个状态，那么状态统计为，这使得 MLE 对于学习 BN 的参数非常的不稳定。而为减少 MLE 过度拟合，使用贝叶斯参数估计实现。

fruit\_bn.fit(fruit\_data, estimator=BayesianEstimator, prior\_type="BDeu") # default equivalent\_sample\_size=5  
for cpd in fruit\_bn.get\_cpds():  
 print(cpd)

+---------------+-----+  
| fruit(apple) | 0.5 |  
+---------------+-----+  
| fruit(banana) | 0.5 |  
+---------------+-----+  
+------------+---------------------+---------------------+---------------------+--------------------+  
| fruit | fruit(apple) | fruit(apple) | fruit(banana) | fruit(banana) |  
+------------+---------------------+---------------------+---------------------+--------------------+  
| size | size(large) | size(small) | size(large) | size(small) |  
+------------+---------------------+---------------------+---------------------+--------------------+  
| tasty(no) | 0.30952380952380953 | 0.38235294117647056 | 0.22413793103448276 | 0.7222222222222222 |  
+------------+---------------------+---------------------+---------------------+--------------------+  
| tasty(yes) | 0.6904761904761905 | 0.6176470588235294 | 0.7758620689655172 | 0.2777777777777778 |  
+------------+---------------------+---------------------+---------------------+--------------------+  
+-------------+----------+  
| size(large) | 0.657895 |  
+-------------+----------+  
| size(small) | 0.342105 |  
+-------------+----------+

### 3.5.4.2 结构学习

#### 1）基于得分（Score-based）的结构估计

* 得分函数（Scoring functions）

基于得分的方法是把结构学习问题作为一个优化问题来解决，而衡量模型和数据之间拟合度的常用得分函数有 BDeu 或 K2 等贝叶斯狄利克雷分数（Bayesian Dirichlet scores）和 BIC (也称为 MDL)等贝叶斯信息准则（ Bayesian Information Criterion）等。基于数据集对每个候选结构打分，得分高的结构为最优结构。下述案例生成了一个假的数据集，包含三个随机变量，因为，因此可知结构为的 v-结构。为了验证BDeu、K2和BIC得分函数，假设模型model1符合 v-结构；而模型model2为。从计算结果得知model1的各得分均高于model2，因此估计为最优结构。

import pandas as pd  
import numpy as np  
from pgmpy.estimators import BDeuScore, K2Score, BicScore,ExhaustiveSearch,HillClimbSearch,PC,MmhcEstimator  
from pgmpy.estimators.CITests import chi\_square  
from pgmpy.models import BayesianNetwork  
  
import warnings  
warnings.filterwarnings('ignore')

# create random data sample with 3 variables, where Z is dependent on X, Y:  
data = pd.DataFrame(np.random.randint(0, 4, size=(5000, 2)), columns=list('XY'))  
data['Z'] = data['X'] + data['Y']  
  
bdeu = BDeuScore(data, equivalent\_sample\_size=5)  
k2 = K2Score(data)  
bic = BicScore(data)  
  
model1 = BayesianNetwork([('X', 'Z'), ('Y', 'Z')]) # X -> Z <- Y  
model2 = BayesianNetwork([('X', 'Z'), ('X', 'Y')]) # Y <- X -> Z  
  
print(f'model1:\nBDeuScore=\t{bdeu.score(model1)}\nK2Score=\t{k2.score(model1)}\nBicScore=\t{bic.score(model1)}')  
print(f'model2:\nBDeuScore=\t{bdeu.score(model2)}\nK2Score=\t{k2.score(model2)}\nBicScore=\t{bic.score(model2)}')

model1:  
BDeuScore= -13938.698111661924  
K2Score= -14329.548959442178  
BicScore= -14294.73527862384  
model2:  
BDeuScore= -20909.028603685794  
K2Score= -20935.843622693803  
BicScore= -20953.069518357206

分解分数，即给出待估计变量的潜在父变量情况下的得分。例如下述示例，给出了随机变量潜在不同父变量的得分，可知当父变量为时有最高得分。

print(bdeu.local\_score('Z', parents=[]))  
print(bdeu.local\_score('Z', parents=['X']))  
print(bdeu.local\_score('Z', parents=['X', 'Y']))

-9209.01947362572  
-6991.892692505713  
-57.120667260368634

* 结构搜索（Search strategies）

在随机变量的数量上，DAG 的搜索空间是超指数的（super-exponential）。对于少量变量节点的情况可以用穷举（完全）搜索ExhaustiveSearch返回得分最高的一个；对于含有大量节点的问题，如果使用穷举法，计算成本较高，因此通常使用启发式搜索策略（heuristic search strategies）估计最优结构。

对于只有3个变量的贝叶斯网络结构估计，使用穷举法总共估计有25个 DAGs，其中的结构得分最高。

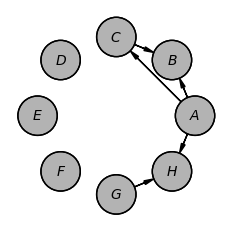
es = ExhaustiveSearch(data, scoring\_method=bic)  
best\_model = es.estimate()  
print(len(list(es.all\_dags())))  
print(best\_model.edges())  
  
print("\nAll DAGs by score:")  
for score, dag in reversed(es.all\_scores()):  
 print(score, dag.edges())

25  
[('X', 'Z'), ('Y', 'Z')]  
  
All DAGs by score:  
-14294.73527862384 [('X', 'Z'), ('Y', 'Z')]  
-14330.662857922875 [('Y', 'X'), ('Z', 'X'), ('Z', 'Y')]  
-14330.662857922875 [('X', 'Z'), ('Y', 'Z'), ('Y', 'X')]  
-14330.662857922875 [('X', 'Y'), ('Z', 'X'), ('Z', 'Y')]  
-14330.662857922875 [('X', 'Y'), ('X', 'Z'), ('Z', 'Y')]  
-14330.662857922875 [('X', 'Y'), ('X', 'Z'), ('Y', 'Z')]  
-14330.662857922876 [('Y', 'Z'), ('Y', 'X'), ('Z', 'X')]  
-16511.124417627434 [('X', 'Y'), ('Z', 'Y')]  
-16513.616802541135 [('Y', 'X'), ('Z', 'X')]  
-18734.18799443991 [('Z', 'X'), ('Z', 'Y')]  
-18734.18799443991 [('X', 'Z'), ('Z', 'Y')]  
-18734.187994439915 [('Y', 'Z'), ('Z', 'X')]  
-20914.64955414447 [('Z', 'Y')]  
-20914.649554144475 [('Y', 'Z')]  
-20917.141939058172 [('Z', 'X')]  
-20917.141939058172 [('X', 'Z')]  
-20950.577133443505 [('Y', 'X'), ('Z', 'Y')]  
-20950.57713344351 [('X', 'Y'), ('Y', 'Z')]  
-20950.577133443512 [('Y', 'Z'), ('Y', 'X')]  
-20953.069518357206 [('X', 'Y'), ('Z', 'X')]  
-20953.069518357206 [('X', 'Y'), ('X', 'Z')]  
-20953.06951835721 [('X', 'Z'), ('Y', 'X')]  
-23097.60349876273 []  
-23133.531078061766 [('Y', 'X')]  
-23133.531078061766 [('X', 'Y')]

如果包含较多的变量节点，则最好使用启发式搜索。例如生成有8个特征列的数据集，其中，，应用爬山算法（hill climbing）计算，从下述结果可以观察到之间，之间出现有向边，而其它随机变量不具有相关性，相互独立，节点之间无边连接。

# create some data with dependencies  
data = pd.DataFrame(np.random.randint(0, 3, size=(2500, 8)), columns=list('ABCDEFGH'))  
data['A'] += data['B'] + data['C']  
data['H'] = data['G'] - data['A']  
  
hc = HillClimbSearch(data)  
best\_model = hc.estimate(scoring\_method=BicScore(data))  
print(best\_model.edges())  
model\_daft=best\_model.to\_daft()  
model\_daft.render(dpi=100);

0%| | 0/1000000 [00:00<?, ?it/s]  
  
  
[('A', 'H'), ('A', 'B'), ('A', 'C'), ('C', 'B'), ('G', 'H')]



#### 2）基于约束（Constrain-based）的结构估计

从数据集中构建 DAG 的一种非常直接的方式是：使用假设检验识别数据集中特征变量的独立性；根据确定的独立性构建 DAG。

* （有条件）独立性检验（（Conditional） Independence Tests）

数据集中特征变量的独立性测试可以使用 chi2 条件独立性测试chi\_square方法识别。例如构建有，关系，下述判断结果满足上述关系，例如之间并不独立，因为之间存在一个节点，使得节点间存在有效迹，节点的影响可以流动到节点；但是如果给定节点，则和之间独立。

data = pd.DataFrame(np.random.randint(0, 3, size=(2500, 8)), columns=list('ABCDEFGH'))  
data['A'] += data['B'] + data['C']  
data['H'] = data['G'] - data['A']  
data['E'] \*= data['F']  
  
print(chi\_square(X='B', Y='H', Z=[], data=data, significance\_level=0.05)) # dependent  
print(chi\_square(X='B', Y='E', Z=[], data=data, significance\_level=0.05)) # independent  
print(chi\_square(X='B', Y='H', Z=['A'], data=data, significance\_level=0.05)) # independent  
print(chi\_square(X='A', Y='G', Z=[], data=data, significance\_level=0.05)) # independent  
print(chi\_square(X='A', Y='G', Z=['H'], data=data, significance\_level=0.05)) # dependent

False  
True  
False  
True  
False

* DAG（模式）构建（DAG (pattern) construction）

借助数据集中特征变量的独立性测试方法，可以分3步从数据集中构建 DAG：

1. 使用build\_skeleton方法构建一个无向骨架（基本结构）（undirected skeleton）；
2. 使用skeleton\_to\_pdag方法定向强制的边获得部分有向无环图（partially directed acyclid graph，PDAG）（DAGs 的 I-等阶类（ I-equivalence class））；
3. 基于 PDAG，使用to\_dag方法保守的定向剩余的边构建 DAG。

步骤1和2使用了称为 PC 的算法[6]。

est = PC(data)  
skel, seperating\_sets = est.build\_skeleton(significance\_level=0.01)  
print("Undirected edges: ", skel.edges())  
  
pdag = est.skeleton\_to\_pdag(skel, seperating\_sets)  
print("PDAG edges: ", pdag.edges())  
  
model = pdag.to\_dag()  
print("DAG edges: ", model.edges())

0%| | 0/5 [00:00<?, ?it/s]  
  
  
Undirected edges: [('A', 'B'), ('A', 'C'), ('A', 'H'), ('E', 'F'), ('G', 'H')]  
PDAG edges: [('G', 'H'), ('C', 'A'), ('A', 'H'), ('B', 'A'), ('F', 'E'), ('E', 'F')]  
DAG edges: [('G', 'H'), ('C', 'A'), ('A', 'H'), ('B', 'A'), ('F', 'E')]

estimate方法可以将上述3个步骤书写为一行代码。

print(est.estimate(significance\_level=0.01).edges())

0%| | 0/5 [00:00<?, ?it/s]  
  
  
[('G', 'H'), ('C', 'A'), ('A', 'H'), ('B', 'A'), ('F', 'E')]

#### 3）混合（Hybrid）结构设计

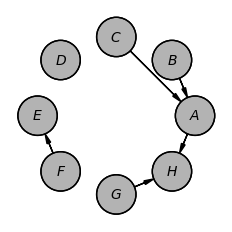
MMHC（Max-Min Hill-Climbing）算法[7]结合了基于约束和基于得分的方法，含有两个步骤：

1. 使用基于约束的结构估计从数据集中学习无向图的基本结构；
2. 使用基于得分的结构估计定向边。

data = pd.DataFrame(np.random.randint(0, 3, size=(2500, 8)), columns=list('ABCDEFGH'))  
data['A'] += data['B'] + data['C']  
data['H'] = data['G'] - data['A']  
data['E'] \*= data['F']  
  
mmhc = MmhcEstimator(data)  
skeleton = mmhc.mmpc()  
print("Part 1) Skeleton: ", skeleton.edges())  
  
# use hill climb search to orient the edges:  
hc = HillClimbSearch(data)  
model = hc.estimate(tabu\_length=10, white\_list=skeleton.to\_directed().edges(), scoring\_method=BDeuScore(data))  
print("Part 2) Model: ", model.edges())

Part 1) Skeleton: [('A', 'C'), ('A', 'B'), ('A', 'H'), ('A', 'E'), ('A', 'F'), ('B', 'G'), ('C', 'H'), ('C', 'D'), ('C', 'F'), ('E', 'F'), ('E', 'G'), ('G', 'H')]  
  
  
  
 0%| | 0/1000000 [00:00<?, ?it/s]  
  
  
Part 2) Model: [('A', 'H'), ('B', 'A'), ('C', 'A'), ('F', 'E'), ('G', 'H')]

model\_daft=model.to\_daft()  
model\_daft.render(dpi=100);

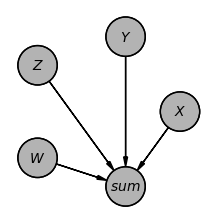


可以分步骤计算，也可以用MmhcEstimator.estimate()方法合并为一个步骤计算。

data = pd.DataFrame(np.random.randint(0, 2, size=(2500, 4)), columns=list('XYZW'))  
data['sum'] = data.sum(axis=1)  
est = MmhcEstimator(data)  
model = est.estimate()  
print(model.edges())

0%| | 0/1000000 [00:00<?, ?it/s]  
  
  
[('X', 'sum'), ('Y', 'sum'), ('Z', 'sum'), ('W', 'sum')]

model\_daft=model.to\_daft()  
model\_daft.render(dpi=100);



### 附：正定（半正定）(Positive (Semi)-Definite)协方差矩阵[8]

协方差（covariance）表示特征向量之间的关系，正协方差表示特征值一起同向变化；负协方差表示特征反向变化。对于两个随机变量有*nm*。

为了方便观察和分类协方差矩阵的变化趋势关系，结合二阶函数几何形态的凹凸性分类有，

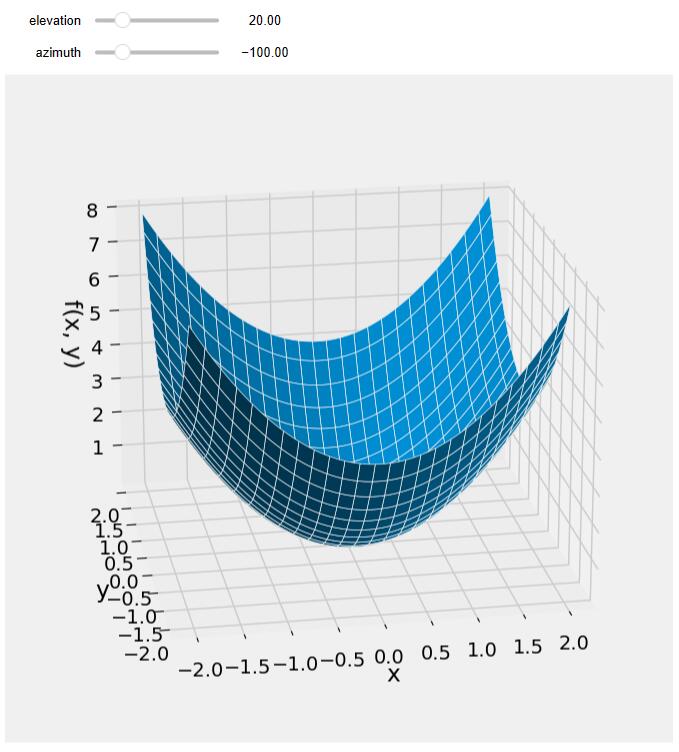
1. 如果一个矩阵是正定的（positive definite），它有一个绝对最小值，且在所有轴上都达到了最小值。如果一个对称矩阵（symmetric matrix）是正定的，满足；
2. 如果一个矩阵是半正定的（positive semi-definite），它至少在一个轴向量上有一个极小值。如果一个对称矩阵（symmetric matrix）是半正定的，满足；
3. 如果一个矩阵是负定的（negative definite ），它有一个绝对最大值，且在所有轴上都达到了最大值；
4. 如果一个矩阵是半负定的（negative semi-definite），它至少在一个轴向量上有一个极大值；
5. 如果一个矩阵是无限的（indfinite），它有一个鞍点。

from datascience import \*  
from prob140 import \*  
%matplotlib inline  
import matplotlib.pyplot as plt  
import numpy as np  
plt.style.use('fivethirtyeight')

%load\_ext autoreload   
%autoreload 2   
import usda.data\_visual as usda\_vis  
import usda.pgm as usda\_pgm

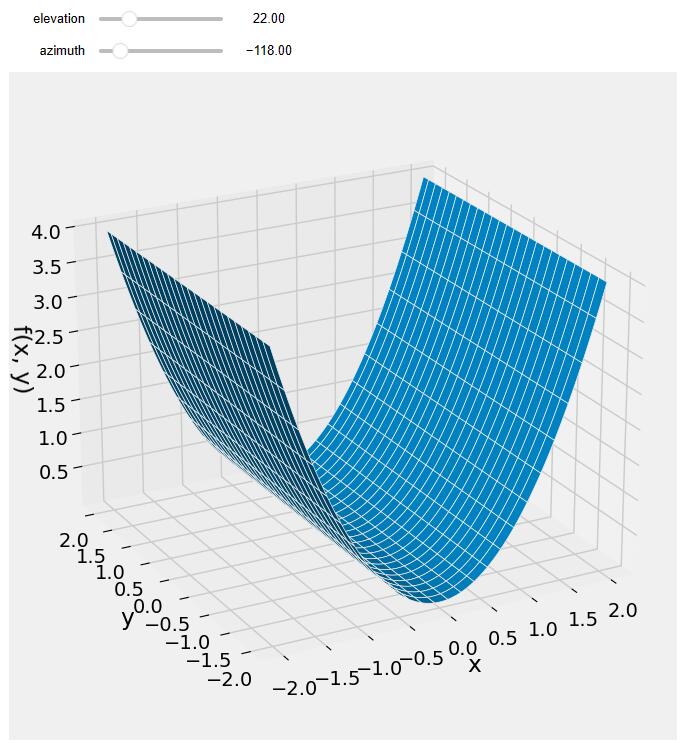
* 正定的（positive definite）

def quadratic\_form\_positive\_definite(x,y):  
 return x\*\*2+y\*\*2  
  
Plot\_3d(x\_limits=(-2,2), y\_limits=(-2,2), f=quadratic\_form\_positive\_definite, cstride=4, rstride=4,interactive=True)



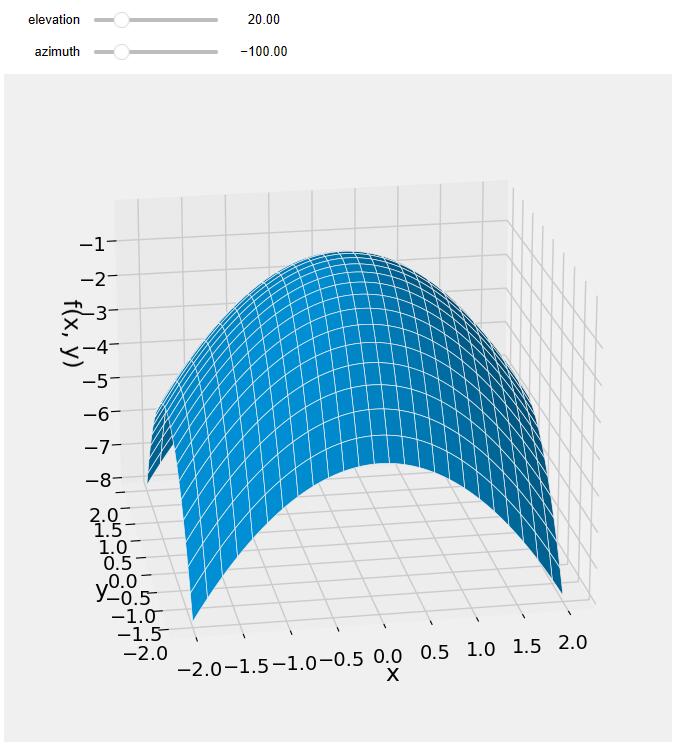
* 半正定的（positive semi-definite）

def quadratic\_form\_positive\_semi\_definite(x,y):  
 return x\*\*2  
  
Plot\_3d(x\_limits=(-2,2), y\_limits=(-2,2),f=quadratic\_form\_positive\_semi\_definite, cstride=4, rstride=4,interactive=True)



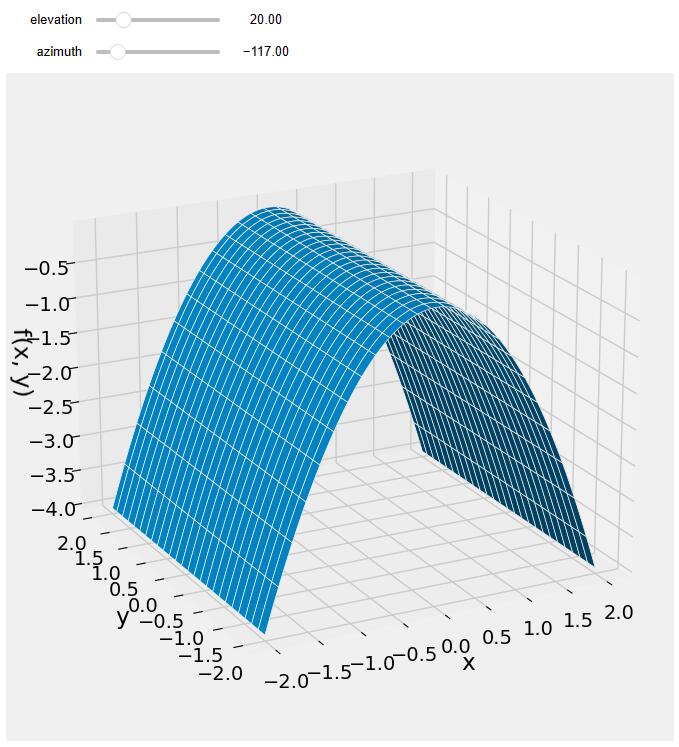
* 负定的（negative definite ）

def quadratic\_form\_negative\_definite(x,y):  
 return -x\*\*2-y\*\*2  
  
Plot\_3d(x\_limits=(-2,2), y\_limits=(-2,2), f=quadratic\_form\_negative\_definite, cstride=4, rstride=4,interactive=True)



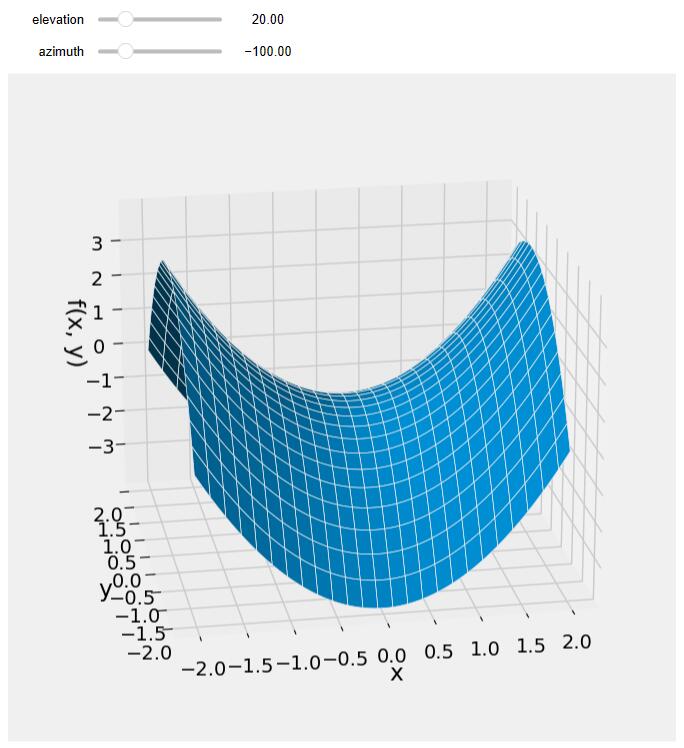
* 半负定的（negative semi-definite）

def quadratic\_form\_negative\_semi\_definite(x,y):  
 return -x\*\*2  
  
Plot\_3d(x\_limits=(-2,2), y\_limits=(-2,2),f=quadratic\_form\_negative\_semi\_definite, cstride=4, rstride=4,interactive=True)



* 无限的（indfinite）

def quadratic\_form\_indefinite(x,y):  
 return x\*\*2-y\*\*2  
  
Plot\_3d(x\_limits=(-2,2), y\_limits=(-2,2),f=quadratic\_form\_indefinite, cstride=4, rstride=4,interactive=True)



使用covariance\_test方法判断协方差矩阵的类别，包括是否是正定、半正定的，是否为对称矩阵，是否为正行列式（Positive Determinant）及特征值（Eigen Values）是否为正。

cov=np.array([[4, 2, -2],   
 [2, 5, -5],   
 [-2, -5, 8]])  
usda\_pgm.covariance\_test(cov,[0,1])

{'Positive Definite': True,  
 'Positive Semi-Definite': True,  
 'Symmetric': True,  
 'Positive Determinant': True,  
 'Eigen Values, Positivity': True}

注释（Notes）：

① pgmpy库， （<https://github.com/pgmpy/pgmpy>）。

② pgmpy在线手册，（<https://pgmpy.org/>）。

③ pomegranate库，（<https://github.com/jmschrei/pomegranate>）。

④ pomegranate在线手册，（<https://github.com/jmschrei/pomegranate/tree/master/docs/tutorials>）。

⑤ PGM\_PyLib库，（<https://github.com/jona2510/PGM_PyLib>）。

⑥ PyGModels库，（<https://github.com/D-K-E/graphical-models/>）。

⑦ PyGModels在线手册，（<https://d-k-e.github.io/graphical-models/html/index.html>）。

⑧ pyGMs库，（<https://github.com/ihler/pyGMs>）。

⑨ pgmPy库，（<https://github.com/indapa/pgmPy>）。

⑩ pgmPy课程，（<https://www.coursera.org/learn/probabilistic-graphical-models>）。

⑪ pgm库，（<https://github.com/paulorauber/pgm>）。

⑫ Skearn（scikit-learn）的GaussianNB类，（<https://scikit-learn.org/stable/modules/naive_bayes.html)>）。

⑬ InVEST（Integrated Valuation of Ecosystem Services and Tradeoffs），（<https://naturalcapitalproject.stanford.edu/software/invest>）。

⑭ daft库，（<https://docs.daft-pgm.org/en/latest/>）。

参考文献（References）:

[1] Koller D, Friedman N. Probabilistic Graphical Models : Principles and Techniques / Daphne Koller and Nir Friedman. MIT Press; 2009.（[美]Daphne Koller [以色列]Nir Friedman 著, 王飞跃、韩素青 译出版社:清华大学出版社出版时间:2015.03）

[2] Ankan A, Panda A. Mastering Probabilistic Graphical Models Using Python : Master Probabilistic Graphical Models by Learning through Real-World Problems and Illustrative Code Examples in Python / Ankur Ankan, Abinash Panda. 1st edition. Packt Publishing; 2015.

[3] Sucar LE. Probabilistic Graphical Models : Principles and Applications / Luis Enrique Sucar. Second edition. Springer; 2021. doi:10.1007/978-3-030-61943-5.

[4] Naive Bayes, <https://scikit-learn.org/stable/modules/naive_bayes.html>.

[5] Heinrich, G. Parameter estimation for text analysis.(2008).

[6] Neapolitan, R. E. Learning Bayesian Networks. (Pearson Prentice Hall, 2004).

[7] Tsamardinos, I., Brown, L. E. & Aliferis, C. F. The max-min hill-climbing Bayesian network structure learning algorithm. Mach Learn 65, 31–78 (2006).

[8] Why Covariance Matrix Should Be Positive Semi-Definite, Tests Using Breast Cancer Dataset，<https://gowrishankar.info/blog/why-covariance-matrix-should-be-positive-semi-definite-tests-using-breast-cancer-dataset/>.