PLS-pathmodeling

2021 年 7 月 2 日

目录

1 模型介绍

偏最小二乘路径分析模型（PLS path modeling）利用 PLS 来估计结构方程模型中系数，常用来对多个变量评分的数据拟合模型，如下所示的消费者满意度的数据，从七个方面 (Image, Expectation, Quality, Value, Satisfaction, Loyalty, Complaints) 来对消费者满意程度进行刻画，而这每个方面在数据中体现为一个或多个可观测变量，由于不可观测，将这七个方面称为隐变量。

[1]:
```
import pandas as pd
import itertools
import pandas as pd
import numpy as np
from pls_pure import pls_pure
from numpy import linalg as nl
import copy

import warnings
warnings.filterwarnings("ignore")
# filterwarnings("error",category=RuntimeWarning)

train = pd.read_csv('mobi.csv', index_col = 0)
x = train.values
train.head()
```

[1]:

	CUEX1	CUEX2	CUEX3	CUSA1	CUSA2	CUSA3	CUSCO	CUSL1	CUSL2	CUSL3	…	\
1	7	7	6	6	4	7	7	6	5	6	…	
2	10	10	9	10	10	8	10	10	2	10	…	
3	7	7	7	8	7	7	6	6	2	7	…	
4	7	10	5	10	10	10	5	10	4	10	…	
5	8	7	10	10	8	8	5	10	3	8	…	

	IMAG5	PERQ1	PERQ2	PERQ3	PERQ4	PERQ5	PERQ6	PERQ7	PERV1	PERV2
1	4	7	6	4	7	6	5	5	2	3
2	9	10	9	10	10	9	10	10	10	10
3	7	7	8	5	7	8	7	7	7	7
4	10	8	10	10	8	4	5	8	5	5
5	9	10	9	8	10	9	9	8	6	6

[5 rows x 24 columns]

偏最小二乘路径分析模型能够刻画每个隐变量的数据与可观测变量之间的关系，以及这七个隐变量之间的结构与因果关系如上图所示，故模型可分为测量模型与结构模型两个部分 (Tenenhaus et al., 2005;

Ray et al., 2021)。

测量模型中，一个隐变量将被显变量块，即一个或多个可观测变量进行表示，其中联系隐变量与显变量的方式主要有三种，分别为 reflective, formative 以及 MIMIC 方式。

结构模型中，通过线性方程来连接各个变量：

$$\boldsymbol{\xi}_j = \beta_{j0} + \sum \beta_{ji}\boldsymbol{\xi}_i + \boldsymbol{v}_i,$$

据此，在上述方程中从未作为因变量出现过的变量称为外生变量，其余变量称之为内生变量，将这些方程联立即可得到隐变量间的关系，称之为结构模型。

[2]:
```
def scale(x, is_scale = True):
    x_bar = np.mean(x, axis = 0)
    x1 = np.copy(x)
    x1 = x1 - x_bar
    if is_scale:
        x_std = np.std(x, axis = 0, ddof = 1)
        x1 = x1 / x_std
        return x1, x_bar, x_std
    return x1, x_bar
```

2 测量模型 (The measurement model)

测量模型中联系显变量与隐变量的方式分为 reflective (mode A), formative (mode B) 与 MIMIC (multiple effect indicators for multiple causes) 三种方式。其中 reflective 与 formative 的方式是最基础的方式，MIMIC 为在前两者基础上进行组合而成的方式，本文仅对前两种方式进行编程。

不论是上述三种方式中的哪一种，将显变量进行中心化处理 $\boldsymbol{y}_j = \boldsymbol{\xi}_j - m_j$ 后，均可表示为

$$\boldsymbol{y}_j \propto \pm\left[\sum w_{jh}(\boldsymbol{x}_{jh} - \bar{x}_{jh})\right],$$

其中 “$\propto$” 表示左式为右式进行标准化处理后的值，“$\pm$” 表示符号的不确定性，可通过选择符号使得 $\boldsymbol{y}_j$ 与大多数 $\boldsymbol{x}_{jh}$ 的符号一致来决定。由于右式还需经过标准化，故不论对显变量 $\boldsymbol{x}_{jh}$ 是否进行标准化都不影响系数的估计，仅需在系数估计值上乘以一个常数即可。则标准化后的隐变量可表示为：

$$\boldsymbol{y}_i = \sum \tilde{w}_{jh}(\boldsymbol{x}_{jh} - \bar{x_{jh}}),$$

w_{jh} 与 $\tilde{w}_{jh}$ 均称为外部权重。$\boldsymbol{\xi}$ 的均值 m_j 由 $\hat{m}_j = \sum \tilde{w}_{jh}\bar{x}_{jh}$ 进行估计，故隐变量的估计形式为：

$$\hat{\boldsymbol{\xi}}_j = \sum \tilde{w}_{jh}\boldsymbol{x}_{jh} = \boldsymbol{y}_j + \hat{m}_j.$$

可将其还原为原有的度量：

$$\hat{\boldsymbol{\xi}}_j^* = \frac{\sum \tilde{w}_{jh}\boldsymbol{x}_{jh}}{\sum \tilde{w}_{jh}},$$

从而对隐变量进行估计。接下来，将依据显变量与隐变量之间的模型形式来确定每次拟合时标准化前的权重 w_{jh}。

2.1 The reflective way

隐变量与显变量间 reflective 的联系方式亦称为 mode A，在这个模型中，显变量是由隐变量映射而成的，每一个隐变量可由其对应显变量表示为如下式所示的简单回归方程：

$$\boldsymbol{x}_h = \pi_{h0} + \pi_h \xi + \boldsymbol{\epsilon_h},$$

其中 $\boldsymbol{x}_h$ 表示显变量，ξ 表示其对应的隐变量，若将该方程中的显变量与隐变量分别进行中心化与标准化的处理后，则可用二者间的相关系数作为外部权重进行估计，

$$w_{jh} = cov(\boldsymbol{x}_{jh}, \boldsymbol{z}_j),$$

$\boldsymbol{z}_j$ 表示标准化处理后的隐变量。

2.2 Code of the reflective way

首先定义 multi_items( ) 与 single_items( ) 函数，表示隐变量所对应的多个显变量。

[3]:
```
def multi_items(item_name, item_numbers):
    items = ['{}{}'.format(item_name, n) for n in range(1, item_numbers+1)]
    return items
def single_items(item):
    return [item]
```

通过 reflective( ) 构建每个隐变量对应显变量块的测量模型，输出结果中"mode" 一列表示该隐变量与显变量之间的 reflective 方式。

[4]:
```
def reflective(construct_name, item_names):
    df = pd.DataFrame([item for item in itertools.product([construct_name],␣
 ↪item_names, "A")], )
#     df = df.values.flatten().reshape(len(item_names), 3)
    df.columns = ["LV", "MV", "mode"]
    return df
```

2.3 The formative way

隐变量与显变量间 formative 的联系方式也可称为 mode B，可以假定因变量是由多个显变量线性组合生成的，即

$$\boldsymbol{\xi} = \sum_h \omega_h \boldsymbol{x}_h + \delta.$$

由于显变量可由隐变量进行回归得来，使用其对应的参数即为该隐变量的外部权重：

$$w_{jh} = (\boldsymbol{X}_j' \boldsymbol{X}_j)^{-1} \boldsymbol{X}_j' \boldsymbol{z}_j$$

上式中的权重是由最小二乘法进行估计得到的权重，而在实际的数据中，显变量间可能具有较严重的多重共线性，OLS 估计量在这种情形下的估计不稳定，若出现完全共线性，则使用最小二乘法无法得到最终的权重估计。进一步地，本文使用 PLS 方法进行估计，从自变量中提取出尽可能多地变量来对因变量进行估计，当自变量不具有完全共线性时，这里所使用的 PLS 估计与最小二乘方法所得到的 OLS 估计值是相同的。

2.4 Code of the formative way

通过 formative( ) 构建每个隐变量对应显变量块的测量模型，输出结果中"mode" 一列表示该隐变量与显变量之间的 reflective 方式。

[5]:
```
def formative(construct_name, item_names):
    df = pd.DataFrame([item for item in itertools.product([construct_name],␣
 ↪item_names, ["B"])], )
    df.columns = ["LV", "MV", "mode"]
#     df = df.values.flatten().reshape(len(item_names),3)
    return df
```

2.5 Code of measurement model

将 reflective( ) 与 formative( ) 所形成的模型结构汇总起来得到所有隐变量的测量模型结构。

[6]:
```
def constructs(*construct_list):
    return_list = pd.concat([*construct_list])
#     way_list = list(map(lambda x: x[0, 2], return_list))
    return return_list
constructs(
    reflective("Image", multi_items("IMAG", 5)),
    formative("Expectation", multi_items("CUEX", 3)),
    reflective("Satisfaction", multi_items("CUSA", 3)),
    formative("Loyalty", multi_items("CUSL", 3)),
    reflective("Quality", multi_items("PERQ", 7)),
    reflective("Complaints", single_items("CUSCO")),
    formative("Value", multi_items("PERV", 2)))
```

[6]:

	LV	MV	mode
0	Image	IMAG1	A
1	Image	IMAG2	A
2	Image	IMAG3	A
3	Image	IMAG4	A
4	Image	IMAG5	A

```
0   Expectation  CUEX1    B
1   Expectation  CUEX2    B
2   Expectation  CUEX3    B
0  Satisfaction  CUSA1    A
1  Satisfaction  CUSA2    A
2  Satisfaction  CUSA3    A
0       Loyalty  CUSL1    B
1       Loyalty  CUSL2    B
2       Loyalty  CUSL3    B
0       Quality  PERQ1    A
1       Quality  PERQ2    A
2       Quality  PERQ3    A
3       Quality  PERQ4    A
4       Quality  PERQ5    A
5       Quality  PERQ6    A
6       Quality  PERQ7    A
0    Complaints  CUSCO    A
0         Value  PERV1    B
1         Value  PERV2    B
```

定义测量模型的类，需要输入显变量的数据（中心化后的），各变量原来的均值以及测量模型结构。该测量模型的类包括两种方法，分别是 scores( ) 与 OuterEstimate( )，其中 scores( ) 能由输入的外部权重与类中所蕴含的测量模型结构得到隐变量的标准化估计得分，OuterEstimate( ) 可以由输入的隐变量得分与类中所蕴含的显变量数据，依据不同的隐变量与显变量的连接方式得到外部权重的初始估计值。

[7]:
```
class MeasurementModel:
    def __init__(self, data, data_mean, MModel):
        self.MModel = MModel
        self.LVs = MModel['LV'].unique().tolist()
        self.MVs = MModel['MV'].unique().tolist()
        self.mean = data_mean
        self.data = data

        self.estimate_way = []
        self.mmMatrix = np.zeros([len(self.MVs), len(self.LVs)])
        for LV in self.LVs:
            MVs_i = MModel.MV[MModel.LV == LV].tolist()
            self.mmMatrix[[self.MVs.index(x) for x in self.MVs if x in MVs_i],
                          self.LVs.index(LV)] = 1
            self.estimate_way.append(MModel[MModel.LV == LV].loc[0,'mode'])
```

```
    def scores(self, outer_weights):
        if type(outer_weights) == pd.core.frame.DataFrame:
            outer_weights = outer_weights.loc[self.MVs, self.LVs]
            outer_weights = outer_weights.values
        scores = np.dot(self.data, outer_weights)
        scores, _, scores_std = scale(scores)
        xty = np.dot(self.data.T, scores)
        sign_weights = []
        for j in range(len(self.LVs)):
            LV = self.LVs[j]
            sign = xty[xty[:, j] > 0, j].size - xty[xty[:, j] <= 0, j].size
            if sign >= 0:
                sign_weights.append(1)
            else:
                sign_weights.append(-1)
        sign_weights = np.array(sign_weights)

        outer_weights = outer_weights * sign_weights
        outer_weights = outer_weights / scores_std

        scores = pd.DataFrame(scores)
        scores.columns = self.LVs
#         scores_bar = np.dot(self.data_bar, outer_weights)
#         scores = scores + scores_bar
        return scores, outer_weights

    def OuterEstimate(self, scores, outer_weights):
        outer_weights_bar = outer_weights.copy()
        LVs = self.LVs
        scores = scores[LVs]
        MModel = self.MModel
        x_scale = self.data
        xtx = np.dot(x_scale.T, x_scale)
        xty = np.dot(x_scale.T, scores)
        x_mean = np.mean(x_scale, axis = 0)
        y_mean = np.mean(scores, axis = 0)
```

```
        n = x_scale.shape[0]
        for j in range(len(self.LVs)):
            LV = LVs[j]
            mode = self.estimate_way[j]
            MVs_j = MModel.MV[MModel.LV == LV].tolist()
            MVs_ind = [self.MVs.index(x) for x in MVs_j]
            xtx_j = xtx[MVs_ind, MVs_ind]
            xty_j = xty[MVs_ind, j]

            pls_bs = pls_pure(xtx_j,
                              xty_j,
                              n = n, p = len(MVs_j))
            if mode == "A":
                outer_weights_bar[MVs_ind, LVs.index(LV)] = pls_bs[0]
            else:
                outer_weights_bar[MVs_ind, LVs.index(LV)] = pls_bs[-1]
        return outer_weights_bar
```

3 结构模型 (The structural model)

3.1 模型介绍

结构模型主要表示隐变量间的关系，内生的隐变量可有其他隐变量的线性组合估计得到。对于标准化内生隐变量 $\boldsymbol{z}_j$ 可由下式进行估计：

$$\boldsymbol{z}_j \propto \sum_{j':\xi_{j'}\ is\ connected\ with\ \xi_j} e_{jj'}\boldsymbol{y}_{j'},$$

只要显变量间存在因果关系，即认为他们之间是有联系的，其中 $e_{jj'}$ 为隐变量间相关系数的符号，这种内部权重的估计方式称为 centroid scheme.

3.2 Code of the structural model

故首先定义隐变量间的因果关系矩阵，paths( ) 关联不同隐变量，需要输入路径的起点与终点，即决定隐变量间的因果关系，relationships( ) 将这些所有的因果关系集合起来，得到结构模型的总体结构。

[8]:
```
def paths(path_from, path_to):
    path = pd.DataFrame([item for item in itertools.product(path_from, path_to)],
                        columns = ["source", "target"])
    return path
```

[9]:
```
def relationships(*path):
    smMatrix = pd.concat([*path])
    return smMatrix
```

由结构模型的总体结构定义结构模型的类，由于隐变量的取值是不可观测的，故对其进行的内部权重估计需要基于测量模型输出的隐变量得分。该类包含一个方法即 InnerEstimate( )，依据测量模型得到的隐变量得分，可通过标准化这些得分，进而使用 centroid scheme 对内部权重进行估计。

[10]:
```
class StructuralModel:
    def __init__(self, SModel):
        self.dependent = SModel['target'].unique().tolist()
        self.construct_vs = SModel.stack().unique().tolist()

#         self.scores = scores[self.construct_vs]
        self.path_Matrix = np.zeros([len(self.construct_vs), len(self.
↪construct_vs)])

        for LV in self.dependent:
            source_i = SModel.source[SModel.target == LV].tolist()
            self.path_Matrix[[self.construct_vs.index(x) for x in self.
↪construct_vs if x in source_i],
                             self.construct_vs.index(LV)] = 1

    def InnerEstimate(self, scores):
        scores = scores[self.construct_vs]
        xtx = np.dot(scores.T, scores)
        inner_weights = np.sign(xtx) * (self.path_Matrix + self.path_Matrix.T)
        scores = np.dot(scores, inner_weights)
        scores, _, estimate_std = scale(scores)
        inner_weights = inner_weights / estimate_std
        scores = pd.DataFrame(scores)
        scores.columns = self.construct_vs
        return scores, inner_weights
```

4 权重估计

4.1 估计方法

本文使用迭代的方法对内部权重、外部权重、隐变量进行估计，即在测量模型中利用外部权重对测量模型中隐变量得分进行估计，再使用所估计得到的得分对结构模型内部权重进行估计，进而通过内部估计

得到隐变量的得分，再由此得分可计算出新的外部权重，从而进行迭代估计，直至外部权重收敛，估计结束。而外部权重的初始值，使用测量模型结构得到的测量模型矩阵如下图所示，每一列表示一个隐变量，每一行表示一个显变量，其中数值为 1 表示该隐变量是由该显变量估计得来的。

4.2 Code of weights estimation

在此定义 PLS path modeling 的类 PLSPathModel，输入所观测到二点显变量数据与测量模型、结构模型的结构，即可通过结构模型与测量模型的迭代估计得到隐变量的估计得分以及内部权重、外部权重的估计值。该类的方法 pls_estimate( ) 需要输入迭代的最大次数以及停止准则，通过外部权重的迭代估计的改变量是否小于停止准则来判断停止迭代。

[11]:
```
class PLSPathModel:
    def __init__(self, data, MModel, SModel):

        self.MVs = MModel['MV'].unique().tolist()
        self.LVs = MModel['LV'].unique().tolist()
        self.data = data[self.MVs]
        x = self.data.values
        self.standard, self.data_bar, self.data_std = scale(x)
        self.centered = x - self.data_bar
        self.n = self.data.shape[0]
        self.mm = MeasurementModel(self.standard, self.data_bar, MModel)
        self.sm = StructuralModel(SModel)

    def pls_estimate(self, maxIt, stopCriterion):
        mm = self.mm
        sm = self.sm

        outer_weights = mm.mmMatrix
        for i in range(maxIt):
            last_outer_weights = outer_weights.copy()
            last_scores, _ = mm.scores(last_outer_weights)
            inner_scores, inner_weights = sm.InnerEstimate(last_scores)
#             try:
#                 outer_weights = mm.OuterEstimate(inner_scores,␣
 ↪last_outer_weights)
#             except RuntimeWarning:
            outer_weights = mm.OuterEstimate(inner_scores, last_outer_weights)
            change = outer_weights - last_outer_weights
            if nl.norm(change[change != 0]) < 10 ** stopCriterion:
```

```
                scores, outer_weights = mm.scores(last_outer_weights)
                scores_bar = np.dot(self.data_bar, outer_weights)
                scores = scores + scores_bar
                inner_weights = pd.DataFrame(inner_weights,
                                             index = sm.construct_vs,
                                             columns =sm.construct_vs)
                outer_weights = pd.DataFrame(outer_weights,
                                             index = mm.MVs,
                                             columns =mm.LVs)
                print("The path model was iterated", i, "times.")
                break
        if i == 299:
            print("The model did not reach convergence.")
        else:
            return scores, outer_weights, inner_weights
```

4.3 函数示例

首先定义结构模型与测量模型的结构，便可输入数据通过 PLSPathModel 这个类得到对隐变量得分、内部权重、外部权重的估计值。如下所定义的模型通过了 14 次迭代后收敛。

[12]:
```
mmModel1 = constructs(
    reflective("Expectation", multi_items("CUEX", 3)),
    reflective("Satisfaction", multi_items("CUSA", 3)),
    reflective("Loyalty", multi_items("CUSL", 3)),
    reflective("Quality", multi_items("PERQ", 7)),
    formative("Value", multi_items("PERV", 2)))

sModel1 = relationships(paths(["Expectation"], ["Quality", "Value",␣
↪"Satisfaction"]),
                        paths(["Quality"], ["Value", "Satisfaction"]),
                        paths(["Value"], ["Satisfaction"]),
                        paths(["Satisfaction"], ["Loyalty"]))
PLSPM = PLSPathModel(train, mmModel1, sModel1)
# PLSPM.pls_estimate(300, -7)
try:
    lantent_vs, out_w, in_w = PLSPM.pls_estimate(300, -7)
    print("The Lantent Variables are\n", lantent_vs)
    print("The outer weights are\n", out_w)
```

```
    print("The inner weights are\n", in_w)
except TypeError:
    print("The model failed to fit.")
```

```
The path model was iterated 14 times.
The Lantent Variables are
      Expectation  Satisfaction    Loyalty    Quality     Value
0       10.205248      7.576419   8.240076   8.590877  0.694845
1       12.597786     10.421098  10.006227  11.732387  0.869354
2       10.416582      8.879838   8.427466   9.543150  0.541084
3       10.807423     10.896778  10.080213   9.914339  0.322238
4       11.365592      9.981768   9.446483  11.197787  0.431661
..            …            …         …         …         …
245     10.416582      7.217856   7.272103   8.716903  2.424871
246     12.011607      9.659556   9.001261  10.910765  0.322238
247     11.002761      9.117678   9.192691  10.168931  0.650508
248     10.145412      8.440508   8.783344   9.353700  0.322238
249     11.210112      9.015131  10.043220  10.997030  3.125749

[250 rows x 5 columns]
The outer weights are
        Expectation  Satisfaction   Loyalty   Quality     Value
CUEX1      0.510871      0.000000  0.000000  0.000000  0.000000
CUEX2      0.486130      0.000000  0.000000  0.000000  0.000000
CUEX3      0.444400      0.000000  0.000000  0.000000  0.000000
CUSA1      0.000000      0.397504  0.000000  0.000000  0.000000
CUSA2      0.000000      0.387762  0.000000  0.000000  0.000000
CUSA3      0.000000      0.415530  0.000000  0.000000  0.000000
CUSL1      0.000000      0.000000  0.454385  0.000000  0.000000
CUSL2      0.000000      0.000000  0.105024  0.000000  0.000000
CUSL3      0.000000      0.000000  0.661912  0.000000  0.000000
PERQ1      0.000000      0.000000  0.000000  0.230940  0.000000
PERQ2      0.000000      0.000000  0.000000  0.161817  0.000000
PERQ3      0.000000      0.000000  0.000000  0.192538  0.000000
PERQ4      0.000000      0.000000  0.000000  0.175451  0.000000
PERQ5      0.000000      0.000000  0.000000  0.177292  0.000000
PERQ6      0.000000      0.000000  0.000000  0.176447  0.000000
PERQ7      0.000000      0.000000  0.000000  0.198177  0.000000
PERV1      0.000000      0.000000  0.000000  0.000000 -1.291313
```

```
PERV2     0.000000      0.000000  0.000000  0.000000  1.291313
The inner weights are
               Expectation   Quality     Value  Satisfaction  Loyalty
Expectation       0.000000  0.472976  0.385556      0.361388      0.0
Quality           0.436495  0.000000  0.385556      0.361388      0.0
Value             0.436495  0.472976  0.000000      0.361388      0.0
Satisfaction      0.436495  0.472976  0.385556      0.000000      1.0
Loyalty           0.000000  0.000000  0.000000      0.361388      0.0
```

参考文献

Ray, S., Danks, N.P., Calero Valdez, A., 2021. seminr: Building and Estimating Structural Equation Models. URL: https://CRAN.R-project.org/package=seminr. r package version 2.0.2.

Tenenhaus, M., Vinzi, V.E., Chatelin, Y.M., Lauro, C., 2005. Pls path modeling. Computational Statistics & Data Analysis 48.