基礎ゼミ 第7回
完全情報最尤法
Full Information Maximum Likelihood

福田研究室 修士1年
平林新

2017年5月22日(月)

アウトライン

- ◆完全情報最尤法とは（おさらい）
- ◆対数尤度関数の構造
- ◆プログラムにおける記述
- ◆様々な推定方法

完全情報最尤法とは

◆仮定を置いて計算を簡略化しない推定方法

	完全情報最尤法	ロジットモデル
分布	正規分布	ガンベル分布
独立性	分散共分散行列	独立

$$U_{1i} = \alpha + \beta X + \varepsilon_{1i}$$
$$U_{2i} = \beta X + \varepsilon_{2i}$$
$$(\varepsilon_{1i}, \varepsilon_{2i}) \sim MVN\left(0, \sum\right)$$
$$\sum = \begin{pmatrix} \sigma_{\varepsilon_1}^2 & \sigma_{\varepsilon_1\varepsilon_2} \\ & 1 \end{pmatrix}$$

※ MVN…多変量正規分布

完全情報最尤法とは

◆内生変数と外生変数

- 内生変数…モデル系内で決定される
  - 運賃や通学時間で決まる定期券ルート
  - 予算や通信速度で決まる通信会社
- 外生変数…モデル系外で決定される
  - 天候に応じた靴の選択
  - 気温に応じた飲料水の選択

完全情報最尤法とは

◆世の中の選択行動を受けて

2つのモデルがお互いを説明変数とするケース

↓

車種選択と自動車利用（走行距離）

↓

選択行動が互いに独立と言えない

↓

完全情報最尤法を利用する

対数尤度関数の構造

◆データの説明

id	type	distance	f_of_w_days	f_of_w_ends
1	1	4625236	20	10
2	1	1200000	20	7
3	2	1030800	5	5
⋮	⋮	⋮	⋮	⋮

id…個人

type…車種（右図）

distance…調査時オドメーターに表示されていた値

f_of_w_days…一ヶ月の平日20日のうち何日運転したか

f_of_w_days…一ヶ月の週末10日のうち何日運転したか

1	ガソリンエンジン・乗用車
2	ガソリンエンジン・軽乗用車
3	ディーゼルエンジン
4	ハイブリッド
5	プラグインハイブリッド
6	電気自動車・乗用車
7	電気自動車・軽自動車
8	その他・乗用車
9	その他・軽乗用車

対数尤度関数の構造

◆定式化

二項離散選択D_n（ガソリン車か否か）

二項離散選択により規定される連続変数Y_n（走行距離）

平日と休日の利用頻度$X_n = {}^t(X_1, X_2)$（回 / 月）

$$Y_n = \alpha + \beta D_n + \varepsilon_n$$

$$Z_n = \gamma \boldsymbol{X}_n + \zeta_n$$

$$D_n = \begin{cases} 1, & if\ Z_n > 0 \\ 0, & otherwise \end{cases}$$

$$(\varepsilon_n, \zeta_n) \sim MVN\left(0, \sum\right), \sum = \begin{pmatrix} \sigma_\varepsilon^2 & \sigma_{\varepsilon\zeta} \\ & 1 \end{pmatrix}$$

$$\rho = \sigma_{\varepsilon\zeta} / \sigma_\varepsilon$$

対数尤度関数の構造

- ◆定式化
  - ◆車種選択と走行距離には相関があると仮定
  - ◆走行距離Y_nは車種選択D_nで説明できると仮定
  - ◆車種選択D_nは乗車頻度で説明できると仮定
  - ◆乗車頻度が多ければ潜在変数Z_nが正の値を取り，D_nが1を取る（個人nはガソリン車を選択する）
  - ◆Y_nとD_nの確率効用に相関があると仮定

対数尤度関数の構造

◆定式化

対数尤度関数LL

$$
\begin{aligned}
&LL(\alpha,\beta,\gamma,\textstyle\sum)\\
&= N\ln\frac{1}{2\pi\sigma_\varepsilon\sqrt{1-\rho^2}}\\
&+\sum_n D_n \ln\int_{-\gamma X_n}^{\infty}\exp\left[-\frac{1}{2(1-\rho^2)}\left\{\left(\frac{Y_n-(\alpha+\beta D_n)}{\sigma_\varepsilon}\right)^2-2\rho\left(\frac{Y_n-(\alpha+\beta D_n)}{\sigma_\varepsilon}\right)t+t^2\right\}\right]dt\\
&+\sum_n (1-D_n)\ln\int_{-\infty}^{-\gamma X_n}\exp\left[-\frac{1}{2(1-\rho^2)}\left\{\left(\frac{Y_n-(\alpha+\beta D_n)}{\sigma_\varepsilon}\right)^2-2\rho\left(\frac{Y_n-(\alpha+\beta D_n)}{\sigma_\varepsilon}\right)t+t^2\right\}\right]dt
\end{aligned}
$$

プログラムにおける記述

◆importしよう
◆csvからデータを取り込もう，このとき型に注意しよう
◆配列を生成しよう
◆range()を使おう
◆スケールを調整しよう
◆np.を使おう
◆関数を定義しよう
◆積分しよう
◆Sumationしよう

プログラムにおける記述

◆importしよう

- ◆解析用の関数
- ◆csvの取り込み
- ◆ランタイムの測定

```
# import the packages
import numpy as np
from scipy.optimize import minimize
import scipy.stats as stats
import time
import pandas as pd
import csv
from scipy import integrate
from scipy.integrate import quad
from sympy import *
import sympy as sp
import time
from scipy import optimize
```

プログラムにおける記述

◆csvからデータを取り込もう，このとき型に注意しよう

◆pandas.core.frame.DataFrameのままでは解析できない

◆np.arrayで計算できる型に変換する

```
#import the data
originaldata = pd.read_csv("mr_abe_data_small.csv")
print type(originaldata)
data = np.array(originaldata)
print type(data)
```

```
<class 'pandas.core.frame.DataFrame'>
<type 'numpy.ndarray'>
```

プログラムにおける記述

◆配列を生成しよう

◆csvからデータを受け取る箱を作る

◆データサイズをlen()で取得して用いると良い

```
# measure the length of data
nrow = len(data)
print nrow
```

```
1000
```

```
# create the variables of distance
yn = np.zeros(nrow)
print yn
```

```
[ 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.
```

```
# create the variables of distance
yn = np.zeros(5)
print yn
```

```
[ 0.  0.  0.  0.  0.]
```

プログラムにおける記述

◆range()を使おう

◆今回，二項離散選択をガソリン車か否か，としている

◆データの変換を行う際，for文をうまく使う必要がある

1	ガソリンエンジン・乗用車
2	ガソリンエンジン・軽乗用車
3	ディーゼルエンジン
4	ハイブリッド
5	プラグインハイブリッド
6	電気自動車・乗用車
7	電気自動車・軽自動車
8	その他・乗用車
9	その他・軽乗用車

```
# put the original data into variables
v_t = data[:,1]# type
v_d = data[:,2] # distance
v_fowd = data[:,3] # frequency of weekdays
v_fowe = data[:,4] # frequency of weekends
```

```
# filter data weather gasoline or not
v_g = np.zeros(nrow) # gasoline
for i in range(0, nrow):# 0 to nrow-1, take care.
    if v_t[i] == 1 or v_t[i] == 2:
        v_g[i] = 1
    else:
        v_g[i] = 0
```

プログラムにおける記述

◆スケールを調整しよう

◆走行距離のデータは他と比べ値が大きい

◆対数を取ることで値同士の差を詰める

◆頻度が0のものの対数は取れないので各値に1を加えて対数を取る

```
# put data into variables
yn = np.log(v_d / (10.0**4)) # Y_n, scaled 1/10,000
dn = v_g # D_n
xdn = np.log(v_fowd + 1) # X frequecy of weekDays
xen = np.log(v_fowe + 1) # X frequecy of weekEnds
```

プログラムにおける記述

◆np.を使おう

◆対数（log）やπ，スクエアルート（sqrt）などを使うのに必要

```
# define the Constant.
def constant(N, s_ee, s_ez):
    value = N * np.log(1 / (2 * np.pi * np.sqrt(s_ee) * np.sqrt(1 - s_ez ** 2 / s_ee) ))
    return value
```

プログラムにおける記述

◆関数を定義しよう

◆引数を決めて返り値を定義する

例）

```
def f(x):
        value = x ** 2
        return value
print f(3)
```

→3^2で9が出力される

```
# the integral function part 1
def integ1(t, n, yn, dn, xdn, xen, b_a, b_b, b_d, b_e, s_ee, s_ez):
    fro = - (b_d * xdn + b_e + xen)
    to = np.inf
    res, err = quad(inexpo, fro, to, args=(n, yn, dn, xdn, xen, b_a, b_b, b_d, b_e, s_ee, s_ez))
    return res
```

プログラムにおける記述

◆積分しよう

◆from scipy.integrate import quad

◆変数tについて積分する場合，他の変数は固定する

```
# the integral function part 1
def integ1(t, n, yn, dn, xdn, xen, b_a, b_b, b_d, b_e, s_ee, s_ez):
    fro = - (b_d * xdn + b_e + xen)
    to = np.inf
    res, err = quad(inexpo, fro, to, args=(n, yn, dn, xdn, xen, b_a, b_b, b_d, b_e, s_ee, s_ez))
    return res
```

プログラムにおける記述

◆Sumationしよう

◆for文と自分で設定した関数の組み合わせ

```
# the sumation method. part 1
def suminteg1(t, N, n, yn, dn, xdn, xen, b_a, b_b, b_d, b_e, s_ee, s_ez):
    result = 0
    for i in range(0, N):# 0 to N-1, take care.
        result += dn[i] * np.log(integ1(t, n, yn[i], dn[i], xdn[i], xen[i], b_a, b_b, b_d, b_e, s_ee, s_ez))#imamadeno
    return result
```

```
# the sumation method. part 2
def suminteg2(t, N, n, yn, dn, xdn, xen, b_a, b_b, b_d, b_e, s_ee, s_ez):
    result = 0
    for i in range(0, N):# 0 to N-1, take care.
        result += (1 - dn[i]) * np.log(integ2(t, n, yn[i], dn[i], xdn[i], xen[i], b_a, b_b, b_d, b_e, s_ee, s_ez))#imamadeno
    return result
```

様々な推定方法

- ◆最適化のMethodは多く存在している
  - ◆ 1次元最適化
    - ◆ scipy.optimize.brent()
  - ◆ 勾配に基づく方法
    - ◆ 最も急な 方向に向かって小さく進んでいく
    - ◆ 急激な変化を持つ関数に対して弱い
  - ◆ Newton法，準Newton法
    - ◆ scipy.optimize.fmin_ncg
    - ◆ **BFGS**
    - ◆ **L-BFGS**
  - ◆ シューティング法
    - ◆ 勾配を利用したアプローチに近い
  - ◆ シンプレックス法
    - ◆ 勾配を計算しないので，急激な変化（ノイズ）にも強い
  - ◆ ブルートフォース（大局的安定）
    - ◆ おおよその最適解の位置の情法がつかめていないとき

様々な推定方法

◆今回用いたのは以下の2種類

◆Nelder-Mead

◆L-BFGS-B

```
[t, N, n] = [1, nrow, 9]
optimize.minimize(LL2, [4.00, 5.000, 0.05, 0.025, 0.8, 0.4], args=(t, N, n, yn, dn, xdn, xen), method = 'Nelder-Mead')
```

```
[t, N, n] = [1, nrow, 9]
res=optimize.minimize(LL2, [4.00, 5.000, 0.05, 0.025, 0.8, 0.4], args=(t, N, n, yn, dn, xdn, xen), method = 'L-BFGS-B')
```

References

◆Wes McKinney(2013)： 『Pythonによるデータ分析入門』， O’REILLY.

◆Scipy Lecture Notes『2.7. 数学的最適化: 関数の最小値を求める』，
http://www.turbare.net/transl/scipy-lecture-notes/advanced/mathematical_optimization/index.html

◆SciPy.org， https://docs.scipy.org/doc/scipy/reference/index.html