因子分析

21C1034 苅込凌太郎

2024 年 7 月 26 日

1　因子分析とは [1–3]

因子分析とは，統計学上のデータ解析手法の一つである．「因子」とは何かの結果を引き起こす原因を意味する．データが持つ複数の要素 (各変数) に共通する因子を探索する分析手法のことである．因子分析は「知能」という潜在的な概念を研究する過程で生まれた分析手法である．これは目に見えず，直接測ることができない「知能」というものが存在し，それが具体的な知能テストや試験などの結果として観測できるという考えをもとにしている．因子分析は主に消費者の行動を理解するために利用されている．

2　プログラム

2.1　変数

資料 [1] ではつぼの遷移確率は「a」，ボールの出力確率は「b」だったが，今回のプログラムではそれぞれ「$moveP$」「$outP$」で宣言している．

ソースコード 1　変数

```
1       //つぼの遷移確率.
2       double moveP[2][2] = {{0.5, 0.5},
3                             {0.5, 0.5}};
4       //ボールの出力確立.
5       double outP[2][2] = {{0.5, 0.5},
6                            {0.5, 0.5}};
7       //どのつぼでどのボールが出力されるかの確率.
8       double caseP[2][3][2] = {{{0, 0}, {0, 0}, {0, 0}},
9                                {{0, 0}, {0, 0}, {0, 0}}};
10      //ボールが出力されるつぼが選ばれる確率.
11      double potP[2][3][2] = {{{0, 0}, {0, 0}, {0, 0}},
12                              {{0, 0}, {0, 0}, {0, 0}}};
13      //つぼがループしたか遷移したかを数えるカウンター.
14      int moveC[3][2][2];
15      //どのボールが出力されたかを数えるカウンター.
16      int outC[3][2][2];
17      //求めたい事象の文字列.
18      char tar[] = "RBBB";
19      //求めたい事象の文字列の文字数.
20      int tarN = sizeof(tar) - 1;
21      //求めたい事象の文字列を数値に変換.
22      int tarV[tarN];
23      //事象のそれぞれの確率.
24      double P[tarN];
```

2.2 関数

2.2.1 事象の確率を求める関数: CalculateP

この関数で事象の確率と $moveC$，$outC$ を数えている．つぼが遷移されるタイミングはつぼ 0 は事象ごとに 1 個ごとにずらし，つぼ 1 は最後の出力が終わったとき．何色のボールが出力されるかは，文字列 tar から読み取っている．

ソースコード 2　CalculateP

```
1   void CalculateP(double a[2][2], double b[2][2], int tarN, int *tarV, double *P, int
        moveC[3][2][2], int outC[3][2][2]){
2     int moveN = 0, moveF = 1;
3     double Ptmp = 1;
4     for(int i = 0; i < tarN - 1; i++){
5       for(int j = 0; j < tarN; j++){
6         //1回目の遷移がされていないとき（つぼ 0）.
7         if(moveF){
8           if(j == moveN){
9             Ptmp *= b[0][tarV[j]] * a[0][1];
10            moveC[i][0][1]++; moveF = 0; //遷移をしたらmoveF を立てる.
11          }
12          else{
13            Ptmp *= b[0][tarV[j]] * a[0][0];
14            moveC[i][0][0]++;
15          }
16          if(tarV[j] == 0) outC[i][0][0]++;
17          else outC[i][0][1]++;
18        }
19        //1回目の遷移がされたとき（つぼ 1）.
20        else{
21          if(j == tarN - 1){
22            Ptmp *= b[1][tarV[j]] * a[1][1];
23            moveC[i][1][1]++;
24          }
25          else{
26            Ptmp *= b[1][tarV[j]] * a[1][0];
27            moveC[i][1][0]++;
28          }
29          if(tarV[j] == 0) outC[i][1][0]++;
30          else outC[i][1][1]++;
31        }
32      }
33      moveN++; moveF = 1;
34      P[i] = Ptmp; Ptmp = 1;
35    }
36  }
```

2.2.2 **遷移確率と出力確率を更新するための確率を求める関数**: CalculateCaseP

この関数でどのつぼでボールが出力されるかの確率 $caseP$ とボールが出力されるつぼが選ばれる確率 $potP$ を計算する．CalculateP で数えた $moveC$，$outC$ を使用して確率をそれぞれ求める．

ソースコード 3　CalculateCaseP

```
1   void CalculateCaseP(int tarN, int C[3][2][2], double caseP[2][3][2]){
2     double sum[3] = {0, 0, 0};
3     for(int n = 0; n < 2; n++){
4       for(int i = 0; i < tarN - 1; i++){
5         sum[i] = (double)(C[i][n][0] + C[i][n][1]);
6         if(C[i][n][0] == 0){caseP[n][i][0] = 0.0; caseP[n][i][1] = 1.0;}
7         else if(C[i][n][1] == 0){caseP[n][i][0] = 1.0; caseP[n][i][1] = 0.0;}
8         else{
9           caseP[n][i][0] = C[i][n][0] / sum[i];
10          caseP[n][i][1] = C[i][n][1] / sum[i];
11        }
12      }
13    }
14  }
```

2.2.3 **遷移確率と出力確率を更新する関数**

HMM の学習のために確率を更新するための関数．事象の確率 P を使用して，$caseP$ と $outP$ を更新する．

ソースコード 4　UpdateP

```
1   void UpdateP(int tarN, double caseP[2][3][2], double P[2], double outP[2][2]){
2     double tmpP[2][2] = {{0, 0}, {0, 0}};
3     double nor = 0;
4     for(int n = 0; n < 2; n++){
5       nor = 0;
6       for(int i = 0; i < 2; i++){
7         for(int j = 0; j < tarN - 1; j++){
8           tmpP[n][i] += P[j] * caseP[n][j][i];
9         }
10        nor += tmpP[n][i];
11      }
12      if(tmpP[n][0] == 0){
13        outP[n][0] = 0.0; outP[n][1] = 1.0;
14      }
15      else if(tmpP[n][1] == 0){
16        outP[n][0] = 1.0; outP[n][1] = 0.0;
17      }
18      else{
19        outP[n][0] = tmpP[n][0] * (1 / nor);
20        outP[n][1] = tmpP[n][1] * (1 / nor);
21      }
22    }
23  }
```

2.2.4 main 関数

学習はそれぞれのつぼの遷移確率と出力確率の更新前と更新後の差が 0.000001 になったら終了.

```
int main(){
  //つぼの遷移確率.
  double moveP[2][2] = {{0.5, 0.5},
                        {0.5, 0.5}};
  //ボールの出力確立.
  double outP[2][2] = {{0.5, 0.5},
                       {0.5, 0.5}};
  //どのつぼでどのボールが出力されるかの確率.
  double caseP[2][3][2] = {{{0, 0}, {0, 0}, {0, 0}},
                           {{0, 0}, {0, 0}, {0, 0}}};
  //ボールが出力されるつぼが選ばれる確率.
  double potP[2][3][2] = {{{0, 0}, {0, 0}, {0, 0}},
                          {{0, 0}, {0, 0}, {0, 0}}};
  //つぼがループしたか遷移したかを数えるカウンター.
  int moveC[3][2][2];
  //どのボールが出力されたかを数えるカウンター.
  int outC[3][2][2];
  //求めたい事象の文字列.
  char tar[] = "RBB";
  //求めたい事象の文字列の文字数.
  int tarN = sizeof(tar) - 1;
  //求めたい事象の文字列を数値に変換.
  int tarV[tarN];
  //事象のそれぞれの確率.
  double P[tarN];

  for(int i = 0; i < tarN; i++){
    if(tar[i] == 'R') tarV[i] = 0;
    else tarV[i] = 1;
  }
  for(int i = 0; i < 3; i++){
    for(int j = 0; j < 2; j++){
      for(int k = 0; k < 2; k++){
        outC[i][j][k] = 0;
        moveC[i][j][k] = 0;
      }
    }
  }

  double tmpO[2][2] = {{10, 10}, {10, 10}};
  double tmpM[2][2] = {{10, 10}, {10, 10}};
  int converN;
  double converV = 0.000001;
  for(int i = 0; i < 100; i++){
    CalculateP(moveP, outP, tarN, tarV, P, moveC, outC);
    CalculateCaseP(tarN, outC, caseP);
    CalculateCaseP(tarN, moveC, potP);
    UpdateP(tarN, caseP, P, outP);
    UpdateP(tarN, potP, P, moveP);
    printf("a = ");
    for(int i = 0; i < 2; i++){
```

```
52              printf("{%lf, %lf}, ", moveP[i][0], moveP[i][1]);
53            }
54            printf("\n");
55            printf("b = ");
56            for(int i = 0; i < 2; i++){
57              printf("{%lf, %lf}, ", outP[i][0], outP[i][1]);
58            }
59            printf("\n");
60            printf("\n");
61            if(fabs(tmpM[0][0]-moveP[0][0]) < converV && fabs(tmpM[0][1]-moveP[0][1]) <
                  converV &&
62               fabs(tmpM[1][0]-moveP[1][0]) < converV && fabs(tmpM[1][1]-moveP[1][1]) <
                      converV &&
63               fabs(tmpO[0][1]-outP[0][1]) < converV && fabs(tmpO[0][1]-outP[0][1]) <
                      converV &&
64               fabs(tmpO[1][0]-outP[1][0]) < converV && fabs(tmpO[1][1]-outP[1][1]) <
                      converV){
65              converN = i+1;
66              printf("学習回数: %d\n", converN);
67              break;
68            }
69            for(int i = 0; i < 2; i++){
70              tmpM[i][0] = moveP[i][0];
71              tmpM[i][1] = moveP[i][1];
72            }
73            for(int i = 0; i < 2; i++){
74              tmpO[i][0] = outP[i][0];
75              tmpO[i][1] = outP[i][1];
76            }
77          }
78
79          return 0;
80        }
```

3　結果

3.1　3 回出力

3 つボールが出力されるとき，2 回目の学習で収束する場合を除くと平均約 6 回で収束した.

3.1.1　**例**)RBB

6 回目の学習で収束した.

```
    $ gcc -lm main.c && ./a.out
    a = {0.250000, 0.750000}, {0.250000, 0.750000},
    b = {0.750000, 0.250000}, {0.000000, 1.000000},

    a = {0.100000, 0.900000}, {0.400000, 0.600000},
    b = {0.900000, 0.100000}, {0.000000, 1.000000},

    a = {0.012195, 0.987805}, {0.487805, 0.512195},
    b = {0.987805, 0.012195}, {0.000000, 1.000000},

    a = {0.000152, 0.999848}, {0.499848, 0.500152},
    b = {0.999848, 0.000152}, {0.000000, 1.000000},

    a = {0.000000, 1.000000}, {0.500000, 0.500000},
    b = {1.000000, 0.000000}, {0.000000, 1.000000},

    a = {0.000000, 1.000000}, {0.500000, 0.500000},
    b = {1.000000, 0.000000}, {0.000000, 1.000000},

    学習回数: 6
```

3.2　4 回出力

(色の組合せ) = (学習回数)

1. すべて同じ色のとき
   - RRRR = 2
   - BBBB = 2
2. 1 つだけ違う色のとき
   - BRRR = 6
   - RBRR = 78
   - RRBR = 78
   - RRRB = 6
   - RBBB = 6
   - BRBB = 78
   - BBRB = 78
   - BBBR = 6
3. 2 つずつ出力されたとき
   - BBRR = 13
   - BRBR = 9
   - BRRB = 2
   - RRBB = 13
   - RBRB = 9
   - RBBR = 2

3.2.1 **例**)RBBB

6 回目の学習で収束した.

```
$ gcc -lm main.c && ./a.out
a = {0.388889, 0.611111}, {0.388889, 0.611111},
b = {0.611111, 0.388889}, {0.000000, 1.000000},

a = {0.191717, 0.808283}, {0.559118, 0.440882},
b = {0.808283, 0.191717}, {0.000000, 1.000000},

a = {0.033409, 0.966591}, {0.653735, 0.346265},
b = {0.966591, 0.033409}, {0.000000, 1.000000},

a = {0.000854, 0.999146}, {0.666381, 0.333619},
b = {0.999146, 0.000854}, {0.000000, 1.000000},

a = {0.000001, 0.999999}, {0.666666, 0.333334},
b = {0.999999, 0.000001}, {0.000000, 1.000000},

a = {0.000000, 1.000000}, {0.666667, 0.333333},
b = {1.000000, 0.000000}, {0.000000, 1.000000},

a = {0.000000, 1.000000}, {0.666667, 0.333333},
b = {1.000000, 0.000000}, {0.000000, 1.000000},

a = {0.000000, 1.000000}, {0.666667, 0.333333},
b = {1.000000, 0.000000}, {0.000000, 1.000000},

a = {0.000000, 1.000000}, {0.666667, 0.333333},
b = {1.000000, 0.000000}, {0.000000, 1.000000},

a = {0.000000, 1.000000}, {0.666667, 0.333333},
b = {1.000000, 0.000000}, {0.000000, 1.000000},

学習回数: 6
```

4　考察

ボールの出力される回数が 3 回のときと 4 回のときを比べると，やはり，4 回のときの方が学習回数が多くなっている．4 回出力されている結果のみに着目すると，同じ色が出力される回数が同じ集団でも学習回数にばらつきが出ているが，対称性が存在していることがわかる．

学習回数の少ないときは，同じ色が隣り合っているときが多いことがわかる．また，RBRB のように交互に色が選ばれるといった規則性があると学習回数が少なくなることが予測される．

例えばボールの出力される回数が 6 回のときは RRRRRR は学習回数が少なく，BBRRBR のような組み合わせは学習回数が多くなることが予測できる．また，RBRBRB のような規則性のある組み合わせは中間程度の学習回数になることも予測できる．

5　ソースコード

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <math.h>
5
6   void CalculateP(double a[2][2], double b[2][2], int tarN, int *tarV, double *P, int
        moveC[3][2][2], int outC[3][2][2]);
7   void CalculateCaseP(int tarN, int C[3][2][2], double P[2][3][2]);
8   void UpdateP(int tarN, double caseP[2][3][2], double P[2], double outP[2][2]);
9
10  int main(){
11    //つぼの遷移確率.
12    double moveP[2][2] = {{0.5, 0.5},
13                          {0.5, 0.5}};
14    //ボールの出力確立.
15    double outP[2][2] = {{0.5, 0.5},
16                         {0.5, 0.5}};
17    //どのつぼでどのボールが出力されるかの確率.
18    double caseP[2][3][2] = {{{0, 0}, {0, 0}, {0, 0}},
19                             {{0, 0}, {0, 0}, {0, 0}}};
20    //ボールが出力されるつぼが選ばれる確率.
21    double potP[2][3][2] = {{{0, 0}, {0, 0}, {0, 0}},
22                            {{0, 0}, {0, 0}, {0, 0}}};
23    //つぼがループしたか遷移したかを数えるカウンター.
24    int moveC[3][2][2];
25    //どのボールが出力されたかを数えるカウンター.
26    int outC[3][2][2];
27    //求めたい事象の文字列.
28    char tar[] = "RBBB";
29    //求めたい事象の文字列の文字数.
30    int tarN = sizeof(tar) - 1;
31    //求めたい事象の文字列を数値に変換.
32    int tarV[tarN];
33    //事象のそれぞれの確率.
34    double P[tarN];
35
36    for(int i = 0; i < tarN; i++){
37      if(tar[i] == 'R') tarV[i] = 0;
38      else tarV[i] = 1;
39    }
40    for(int i = 0; i < 3; i++){
```

```
41          for(int j = 0; j < 2; j++){
42            for(int k = 0; k < 2; k++){
43              outC[i][j][k] = 0;
44              moveC[i][j][k] = 0;
45            }
46          }
47        }
48
49        double tmpO[2][2] = {{10, 10}, {10, 10}};
50        double tmpM[2][2] = {{10, 10}, {10, 10}};
51        int converN;
52        double converV = 0.000001;
53        for(int i = 0; i < 100; i++){
54          CalculateP(moveP, outP, tarN, tarV, P, moveC, outC);
55          CalculateCaseP(tarN, outC, caseP);
56          CalculateCaseP(tarN, moveC, potP);
57          UpdateP(tarN, caseP, P, outP);
58          UpdateP(tarN, potP, P, moveP);
59          printf("a = ");
60          for(int i = 0; i < 2; i++){
61            printf("{%lf, %lf}, ", moveP[i][0], moveP[i][1]);
62          }
63          printf("\n");
64          printf("b = ");
65          for(int i = 0; i < 2; i++){
66            printf("{%lf, %lf}, ", outP[i][0], outP[i][1]);
67          }
68          printf("\n");
69          printf("\n");
70          if(fabs(tmpM[0][0]-moveP[0][0]) < converV && fabs(tmpM[0][1]-moveP[0][1]) <
                converV &&
71             fabs(tmpM[1][0]-moveP[1][0]) < converV && fabs(tmpM[1][1]-moveP[1][1]) <
                   converV &&
72             fabs(tmpO[0][1]-outP[0][1]) < converV && fabs(tmpO[0][1]-outP[0][1]) < converV
                    &&
73             fabs(tmpO[1][0]-outP[1][0]) < converV && fabs(tmpO[1][1]-outP[1][1]) < converV
                   ){
74            converN = i+1;
75            printf("学習回数: %d\n", converN);
76            break;
77          }
78          for(int i = 0; i < 2; i++){
79            tmpM[i][0] = moveP[i][0];
80            tmpM[i][1] = moveP[i][1];
81          }
82          for(int i = 0; i < 2; i++){
83            tmpO[i][0] = outP[i][0];
84            tmpO[i][1] = outP[i][1];
85          }
86        }
87
88        return 0;
89      }
90
```

```
91      void CalculateP(double a[2][2], double b[2][2], int tarN, int *tarV, double *P, int
            moveC[3][2][2], int outC[3][2][2]){
92        int moveN = 0, moveF = 1;
93        double Ptmp = 1;
94        for(int i = 0; i < tarN - 1; i++){
95          for(int j = 0; j < tarN; j++){
96            //1回目の遷移がされていないとき（壺 0）.
97            if(moveF){
98              if(j == moveN){
99                Ptmp *= b[0][tarV[j]] * a[0][1];
100               moveC[i][0][1]++; moveF = 0; //遷移をしたらmoveF を立てる.
101             }
102             else{
103               Ptmp *= b[0][tarV[j]] * a[0][0];
104               moveC[i][0][0]++;
105             }
106             if(tarV[j] == 0) outC[i][0][0]++;
107             else outC[i][0][1]++;
108           }
109           //1回目の遷移がされたとき（壺 1）.
110           else{
111             if(j == tarN - 1){
112               Ptmp *= b[1][tarV[j]] * a[1][1];
113               moveC[i][1][1]++;
114             }
115             else{
116               Ptmp *= b[1][tarV[j]] * a[1][0];
117               moveC[i][1][0]++;
118             }
119             if(tarV[j] == 0) outC[i][1][0]++;
120             else outC[i][1][1]++;
121           }
122         }
123         moveN++; moveF = 1;
124         P[i] = Ptmp; Ptmp = 1;
125       }
126     }
127
128     void CalculateCaseP(int tarN, int C[3][2][2], double caseP[2][3][2]){
129       double sum[3] = {0, 0, 0};
130       for(int n = 0; n < 2; n++){
131         for(int i = 0; i < tarN - 1; i++){
132           sum[i] = (double)(C[i][n][0] + C[i][n][1]);
133           if(C[i][n][0] == 0){caseP[n][i][0] = 0.0; caseP[n][i][1] = 1.0;}
134           else if(C[i][n][1] == 0){caseP[n][i][0] = 1.0; caseP[n][i][1] = 0.0;}
135           else{
136             caseP[n][i][0] = C[i][n][0] / sum[i];
137             caseP[n][i][1] = C[i][n][1] / sum[i];
138           }
139         }
140       }
141     }
142
143     void UpdateP(int tarN, double caseP[2][3][2], double P[2], double outP[2][2]){
```

```
        double tmpP[2][2] = {{0, 0}, {0, 0}};
        double nor = 0;
        for(int n = 0; n < 2; n++){
          nor = 0;
          for(int i = 0; i < 2; i++){
            for(int j = 0; j < tarN - 1; j++){
              tmpP[n][i] += P[j] * caseP[n][j][i];
            }
            nor += tmpP[n][i];
          }
          if(tmpP[n][0] == 0){
            outP[n][0] = 0.0; outP[n][1] = 1.0;
          }
          else if(tmpP[n][1] == 0){
            outP[n][0] = 1.0; outP[n][1] = 0.0;
          }
          else{
            outP[n][0] = tmpP[n][0] * (1 / nor);
            outP[n][1] = tmpP[n][1] * (1 / nor);
          }
        }
      }
```

参考文献

[1] 株式会社インテージ. ”因子分析とは”. (https://www.intage.co.jp/glossary/050/), (参照日: 2024/07/26)

[2] docomo business. ”因子分析のメリットや結果の見方を具体例から解説”. (https://www.nttcoms.com/service/research/dataanalysis/factor-analysis/), (参照日: 2024/07/26)

[3] MACROMILL. ”因子分析”. (https://www.macromill.com/service/data-analysis/factor-analysis/?uuid=20240723_65baf098_669b_45fe_be69_60f5e7ede0cd), (参照日: 2024/07/26)