スタートアップゼミ
第一回発表

2018/4/30

#3 スタートアップゼミ

B4　飯塚卓哉

ヘッセ行列

実数地関数$f(x_1, x_2, \ldots, x_n)$ にすべての二階偏微分が存在するとき

$$H(f) = \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

で与えられる行列を**ヘッセ行列**という.

ヘッセ行列の行列式を**Hessian**という.

推定値 θ の母分散共分散行列の推定値はヘッセ行列に推定値 θ を代入した値に等しくなる.
→これを利用し，推定値 θ の推定標準偏差を求め，**t値**を求める.

→ パラメータの検定

MNL推定-松山PT-

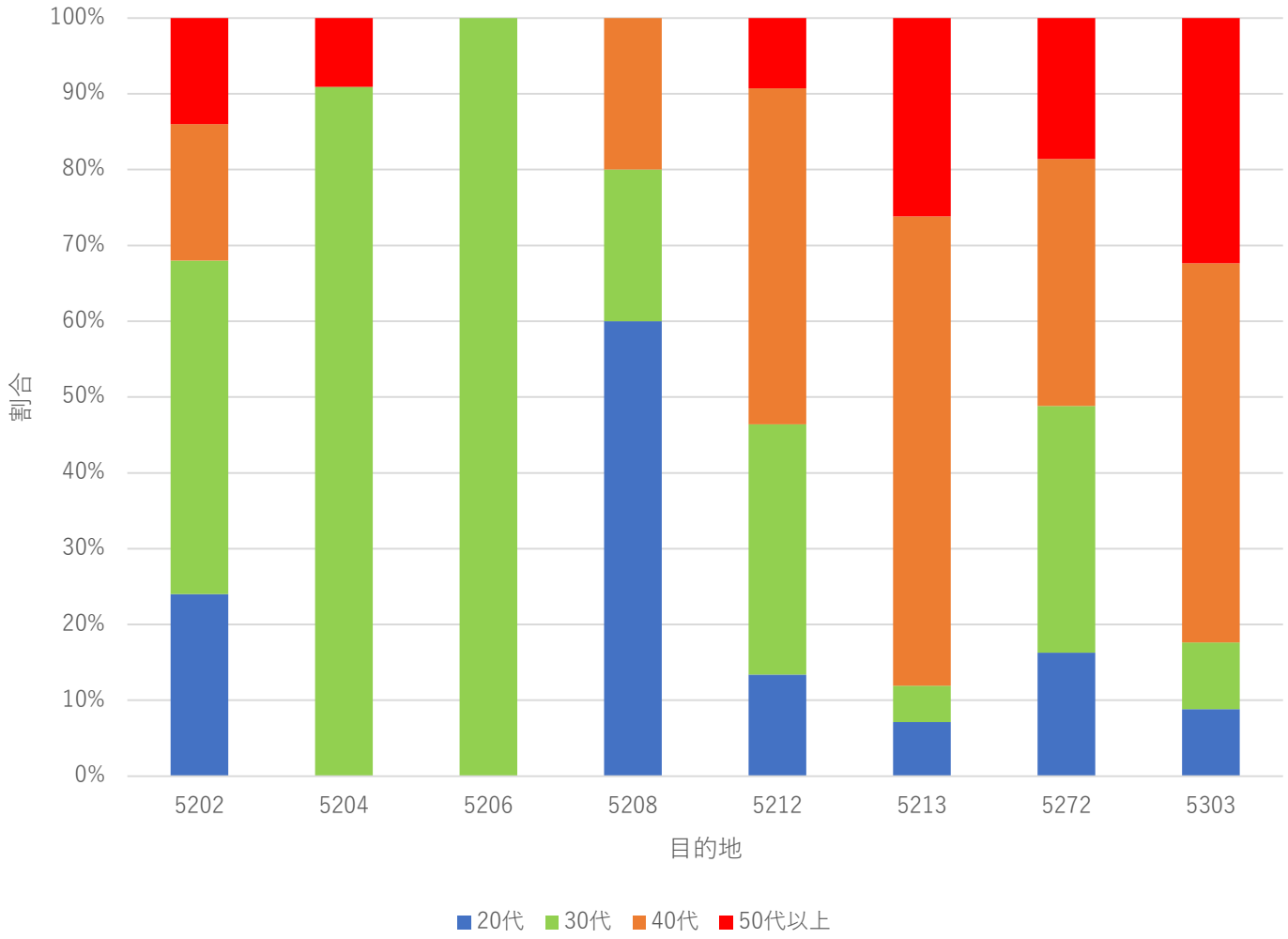
目的地選択モデルのパラメータ推定

選択集合

5202	ﾌｼﾞｸﾞﾗﾝ松山店
5204	ﾌｼﾞｸﾞﾗﾝ重信SC
5206	ﾊﾟﾙﾃｨ姫原SC
5208	ﾊﾟﾙﾃｨ衣山SC
5212	伊予鉄高島屋
5213	三越　松山店
5272	ジャスコ松山店
5303	ジョー・プラ

8項選択の目的地選択モデルを考える

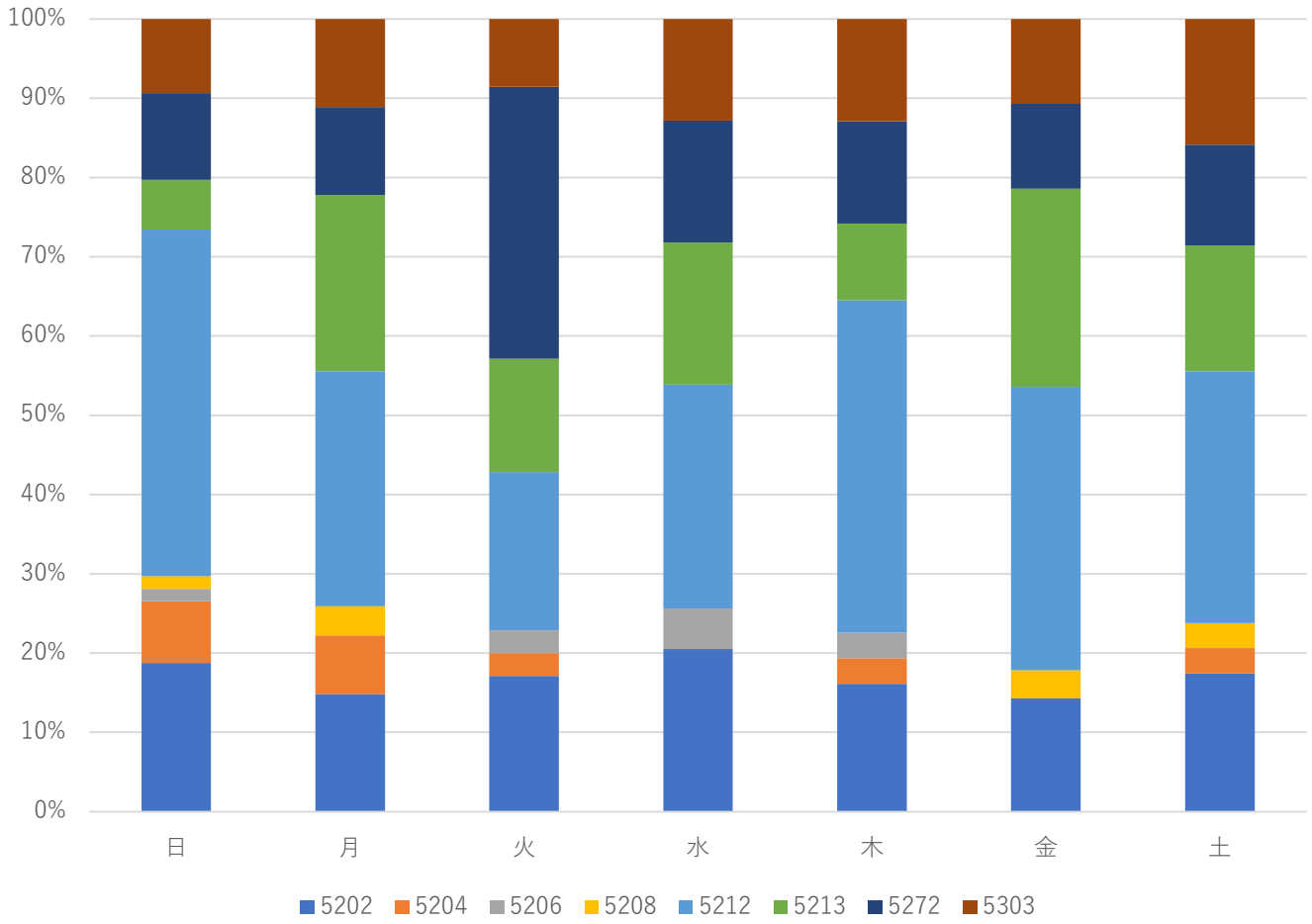
クロス集計 - 目的地別年齢構成 -



伊予鉄高島屋
三越　松山店
ジャスコ松山店
ジョー・プラ

の4つが40代以上の割合が高い

クロス集計 - 曜日別 -



火曜日にジャスコ松山店に行く人の割合が高い

効用関数

$$
\begin{aligned}
V_{5202} &= d_1X_1 && + m_1X_3 + p_1\delta_1 && + \beta_1 \\
V_{5204} &= d_1X_1 && + m_1X_3 + p_1\delta_1 && + \beta_2 \\
V_{5206} &= d_1X_1 && + m_1X_3 + p_1\delta_1 && + \beta_3 \\
V_{5208} &= d_1X_1 && + m_1X_3 + p_1\delta_1 && + \beta_4 \\
V_{5212} &= d_1X_1 + a_1X_2 && + m_1X_3 + p_1\delta_1 && + \beta_5 \\
V_{5213} &= d_1X_1 + a_1X_2 && + m_1X_3 + p_1\delta_1 && + \beta_6 \\
V_{5272} &= d_1X_1 + a_1X_2 && + m_1X_3 + p_1\delta_1 + y_1\delta_2 && + \beta_7 \\
V_{5303} &= d_1X_1 + a_1X_2 && + m_1X_3 + p_1\delta_1 &&
\end{aligned}
$$

距離	年齢	売り場面積	駐車場料金	曜日	定数項
m/1000		m2/1000	の有無ダミー	(火曜日ダミー)	

パラメータ推定

説明変数	パラメータ	t値	
距離[m/1000]	-0.8322	-10.6782	**
年齢	0.0301	1.7663	
売り場面積[m2/1000]	0.0325	0.0000	
駐車場料金有無ダミー	0.1593	0.0000	
火曜日ダミー	1.7567	3.8744	**
定数項5202	1.3032	0.0003	
定数項5204	-0.3711	-0.0002	
定数項5206	-0.3441	-0.0001	
定数項5208	-0.8551	-0.0001	
定数項5212	0.4405	0.0001	
定数項5213	-0.2813	-0.0001	
定数項5272	-0.1322	-0.0002	
サンプル数	287		
初期尤度	-596.7997		
最終尤度	-393.5241		
尤度比	0.3406		
修正済み尤度比	0.3205		

*5%有意　**1%有意

移動手段=徒歩の場合

・効用関数から売り場面積，駐車場料金有無の項を省く
・5.0km以上歩かなければならない選択肢は省く

説明変数	パラメータ	t値
距離[m/1000]	-2.4233	-6.9356**
年齢	0.0363	0.5742
火曜日ダミー	1.1141	0.8976
定数項5202	0.6179	0.245
定数項5204	-5.1155	-0.0109
定数項5206	-6.5905	-0.0727
定数項5208	-8.997	-0.0795
定数項5212	0.0419	0.0529
定数項5213	-0.575	-0.6935
定数項5272	-1.1348	-1.8202
サンプル数	98	
初期尤度	-175.5894	
最終尤度	-65.5012	
尤度比	0.627	
修正済み尤度比	0.57	

*5%有意　**1%有意

移動手段=自動車の場合

・効用関数は徒歩の場合と同じ
・距離による選択肢の限定は無し

説明変数	パラメータ	t値
距離[m/1000]	-0.4980	-5.7682**
年齢	0.0255	1.0970
火曜日ダミー	1.9866	3.0413**
定数項5202	1.3711	1.4133
定数項5204	0.1532	0.1331
定数項5206	-1.5444	-1.1149
定数項5208	-1.1586	-0.9800
定数項5212	0.6473	2.2406*
定数項5213	-0.6322	-1.6364
定数項5272	0.0008	0.0024
サンプル数	129	
初期尤度	-268.2480	
最終尤度	-195.5283	
尤度比	0.2711	
修正済み尤度比	0.2338	

*5%有意　**1%有意

考察

- 移動手段別にセグメンテーションしない場合，距離と火曜日ダミーが有意.

- 徒歩で移動する人については，遠すぎる選択肢を省いても距離が有意となり，モデルの尤度比が高くなる.

  →徒歩で移動する人については距離が非常に説明力の高い変数となる.

- 自動車で移動する人については，モデルの尤度比が低い.

  →距離，火曜日ダミー以外の要因が目的地選択に影響を与えている可能性がある.

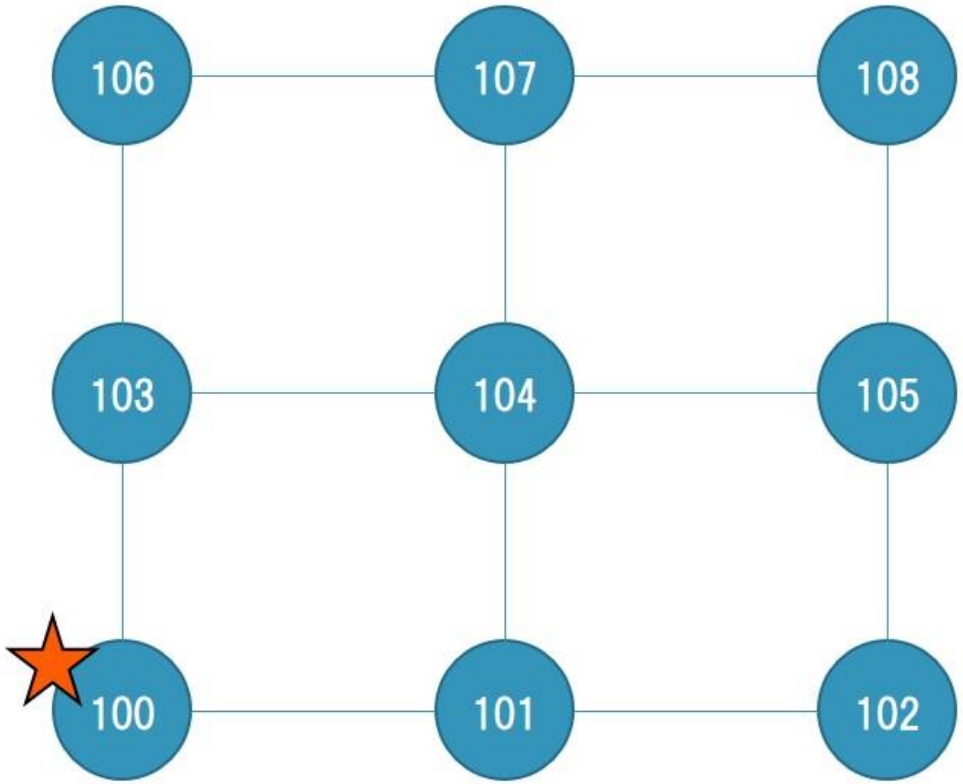
  →自動車で移動する人について他の個人属性を説明変数として導入する，目的地間の相関を考慮したモデルを用いるなどによりモデルを改善させられると思われる.

  （NLモデルに挑戦しましたが計算が回らなかったので引き続き勉強します）

Dijkstra法による最短経路探索と仮想ネットワークにおけるMNL推定

選択集合

発表課題　データ概要



・円内の番号はNodeID(/ArealD)を表す
・現在★の位置にいて他のエリアに行く場合の目的地選択を考えます
・その他の詳細はSampleDataのzip内にあるREADME.txtを参照

ノード100から各ノードまでの最短経路と
そのコストを最短経路探索によって求める

目的地別平均年齢

目的地	トリップ数	平均年齢	スケール	最短距離
101	14	52.1	5	142
102	3	34.3	5	457
103	7	56.4	2	89.7
104	14	40.3	8	347
105	1	24.0	2	538
106	5	48.0	8	464
107	1	23.0	5	703
108	5	29.8	8	832
	50	44.5		

年齢が低いほど距離を重視せずスケールを重視し，
高いと逆にスケールを重視せず，距離の近さを重視する傾向

効用関数

$$
\begin{aligned}
V_{101} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2 + \beta_1 \\
V_{102} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2 + \beta_2 \\
V_{103} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2 + \beta_3 \\
V_{104} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2 + \beta_4 \\
V_{105} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2 + \beta_5 \\
V_{106} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2 + \beta_6 \\
V_{107} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2 + \beta_7 \\
V_{108} &= \delta_y d_1 X_1 + \delta_o d_2 X_1 + \delta_y s_1 X_2 + \delta_o s_2 X_2
\end{aligned}
$$

$\delta_y d_1 X_1$	$\delta_o d_2 X_1$	$\delta_y s_1 X_2$	$\delta_o s_2 X_2$	β
距離(46歳未満)	距離(46歳以上)	スケール(46歳未満)	スケール(46歳以上)	定数項

上式では有意なパラメータは無し

パラメータ推定(46歳未満)

年齢が高いほどスケールを重視しないと考えられるため，スケール(46歳以上)の項を全て削除.

説明変数	パラメータ	t値
距離（age < 46）	5.4485	0.0005
距離（age ≧ 46）	-11.0690	-0.0011
スケール（age < 46）	-0.3136	-0.7209
定数項101	3.2724	0.0005
定数項102	0.5888	0.0002
定数項103	1.9560	0.0003
定数項104	3.6064	0.0007
定数項105	-1.8900	-0.0006
定数項106	1.9973	0.0005
定数項107	-1.8465	-0.0014
サンプル数	50.0000	
初期尤度	-103.9721	
最終尤度	-81.6928	
尤度比	0.2143	
修正済み尤度比	0.1181	

*5%有意　**1%有意

パラメータ推定(46歳以上)

年齢が低いほど距離を重視しないと考えられるため，距離(46歳未満)の項を全て削除.

説明変数	パラメータ	t値
距離（age ≧ 46）	-0.6739	-2.5229*
スケール（age < 46）	0.2803	0.0001
スケール（age ≧ 46）	0.2717	0.0001
定数項101	0.3930	0.0000
定数項102	0.0092	0.0000
定数項103	0.2334	0.0000
定数項104	0.4391	0.7836
定数項105	-0.1195	0.0000
定数項106	-0.2998	-0.4666
定数項107	-0.8138	-0.0001
サンプル数	50.0000	
初期尤度	-103.9721	
最終尤度	-80.5222	
尤度比	0.2255	
修正済み尤度比	0.1294	

*5%有意　**1%有意

考察

- 46歳以上の人について距離が有意に効いている．

  →年齢が高いほど距離が有意に働くと言える．

- 46歳未満の人については有意なパラメータは無し．スケールの有意度は不十分．

  →データ数が少ない，距離やスケールに関係のない目的地選択をするサンプルが多いなどが考えられる．

Dijkstra法の高速化

Heap構造を用いたアルゴリズム

Dijkstra法の高速化

Dijkstra法は，探索点を決定するところがボトルネックとなる．
→未確定ノードの集合を部分的最小費用を優先度とする優先度付きキューとすることで高速化できる

```
def dijkstra(m, start, goal=None):

    num = len(m)  # グラフのノード数

    dist = [float('inf') for i in range(num)]  # 始点から各頂点までの最短距離を格納する

    prev = [float('inf') for i in range(num)]  # 最短経路における，その頂点の前の頂点のIDを格納する

    dist[start] = 0

    q = []  # プライオリティキュー．各要素は，(startからある頂点vまでの仮の距離，頂点vのID)からなるタプル

    heapq.heappush(q, (0, start))  # 始点をpush

    while len(q) != 0:

        prov_cost, src = heapq.heappop(q)  # pop

        # プライオリティキューに格納されている最短距離が，現在計算できている最短距離より大きければ，distの更新をする必要はない

        if dist[src] < prov_cost:
            continue

        # 他の頂点の探索

        for dest in range(num):

            cost = m[src][dest]

            if cost != float('inf') and dist[dest] > dist[src] + cost:
```

Dijkstra法の高速化

```
                dist[dest] = dist[src] + cost  # distの更新
                heapq.heappush(q, (dist[dest], dest))  # キューに新たな仮の距離の情報をpush
                prev[dest] = src  # 前の頂点を記録
    if goal is not None:
        return (get_path(goal, prev), dist[goal])
    else:
        return dist
def get_path(goal, prev):
    path = [goal]  # 最短経路
    dest = goal
    # 終点から最短経路を逆順に辿る
    while prev[dest] != float('inf'):
        path.append(prev[dest])
        dest = prev[dest]
    # 経路をreverseして出力
    return list(reversed(path))
```

```
    dest = goal
    # 終点から最短経路を逆順に辿る
    while prev[dest] != float('inf'):
        path.append(prev[dest])
        dest = prev[dest]
    # 経路をreverseして出力
    return list(reversed(path))

for v in range(len(Olist)):
    origin = int(Olist[v])
    destination = int(Dlist[v])
    print(origin, destination)

    print(dijkstra(m, origin, destination))
```