«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА»

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

Теоретико-возможностные методы анализа и интерпретации данных измерений

Выполнил студент
435 группы:
Голев
Александр Сергеевич
Преподаватель: к.ф.-м.н.
Фаломкина Олеся Владимировна

Москва — 2023

Содержание

1 Постановка задачи

Рассмотрим задачу принятия решения на примере управления автомобилем. Предположим, что нам известны возможные состояния светофора на перекрестке.
$\theta = \{\theta_1, \theta_2, \theta_3\}$

1. Горит зеленый свет светофора.
2. Мигает желтый свет светофора.
3. Горит красный свет светофора.

Также, определены возможные действия. D $= \{d_1, d_2, d_3, d_4, d_5, d_6\}$

1. Проехать перекресток, не сбрасывая скорость.
2. Сбросить скорость до 50 км\ч .
3. Сбросить скорость до 40 км\ч .
4. Сбросить скорость до 20 км\ч .
5. Остановиться, чтобы уступить дорогу, и продолжить движение.
6. Полностью остановиться.

Задана матрица, которая оценивает возможные потери $l(\theta_i, d_j)$ при принятии определенного решения при конкретном состоянии светофора.

Рис. 1: Матрица потерь

	d_1	d_2	d_3	d_4	d_5	d_6
θ_1	1	3	4	6	7	9
θ_2	4	5	5	4	2	9
θ_3	7	6	2	5	3	9

2 Минимаксное и рандомизированное минимаксное правила решения

Мы предполагаем, что не знаем точно, в каком состоянии находился светофор в момент принятия решения. Однако, поскольку мы знаем множество возможных его состояний θ, мы можем определить оптимальное правило как решение задачи минимизирующее максимальный риск, отвечающий наиболее неблагоприятному состоянию светофора.

$$c^* = \min_{d \in D} \max_{\theta \in \Theta} l(\theta, d)$$

Если минимум $l(\theta, d)$ достигается на нескольких действиях $d \in D$, то можем выполнить любое.

```
import numpy as np

matrix = np.array([[1,3,4,6,7,9],[4,5,5,4,2,9],[7,6,2,5,3,9]]).T
min_max_risk = matrix.max(axis=1).min() # min_max_risk = 5
action_index = np.where(matrix == min_max_risk)[0][0] # action_index = 0

print(min_max_risk,action_index+1,points[action_index]) # 5 for action d2. [3 5 6]
```

Итак наше минимаксный риск $c^* = 5$ для действия d_2.

Также можем воспользоваться **рандомизированным правилом** - рассмотрим экспериментом с m случайными исходами $\alpha_1, ..., \alpha_m$, $\sum_i^m p(\alpha_i) = 1$.

Прежде чем принять решение о действии в состоянии светофора θ, можно разыграть случайный эксперимент и принять решение о действии d_{i_p}, если исходом эксперимента окажется α_p.

$$c_r^* = \min\{\max\{l_1(p), ..., l_k(p)\}\}, p \in \mathcal{P}$$

Для нахождения рандомизированного риска я решил выбрать графический подход, однако из-за трехмерности задачи, пришлось решать задачу программно.

Если мы решаем задачу графически, то нам нужно построить график и нарисовать на нём все точки. Затем построить выпуклую оболочку.

```
  import cdd as pcdd

  def get_clean_points(matrix): # cleanup action return function
    N = len(matrix)
    vertices = np.hstack((np.ones((N,1)), matrix)) # to get the convex hull with cdd one
                                                   needs to add a unit column
    mat = pcdd.Matrix(vertices, linear=False, number_type="fraction") # make a polyhedron
    mat.rep_type = pcdd.RepType.GENERATOR
    poly = pcdd.Polyhedron(mat)
    adjacencies = [list(x) for x in poly.get_input_adjacency()] # make a polyhedron
    return [index for index, vert in enumerate(adjacencies) if len(vert)!=0], adjacencies
```

Одним из способов является проведение биссектрисы, вторым - расширение кубической области. Первая точка пересечения с выпуклой областью будет нашим оптимальным рандомизированным риском.

```
  def find_intersection(clean_points, pts,adj):
    intersections = []
    for i in list(combinations(clean_points, 3)):
    # look only at the planes forming a convex hull
      if (i[1] in adj[i[0]] and i[2] in adj[i[0]] and i[1] in adj[i[2]]):
        # intersection point of a plane of three points and a bisector
        intersect = intersect3D_SegmentPlane(LineString([[0,0,0],[10,10,10]]), pts[i,:])
        if (in_poly_hull_single(pts, intersect)): # checking that a point is in polygon
          intersections.append(list(intersect))
    intersections =   np.unique(intersections,axis = 0)
    if len(intersections)>=2:
      intersections = np.array([intersections[0],intersections[-1]])
    return intersections
```

```
def find_random_square(points, ax,min_max_risk):
  vertices = np.hstack((np.ones((len(points),1)), points))
  h1 = Polyhedron(Matrix(vertices, number_type="fraction")).get_inequalities()
  steps = np.linspace(min_max_risk,0.1,5000)
  for index, step in enumarate(steps):
    cube = [(step, 0, 0), (step, step, 0), (step, step,step), (step, 0,step),(0, step, 0)
                                          ,(0, step,step),(0, 0, step), (step, 0, step
                                          )]
    vertices = np.hstack((np.ones((len(cube),1)), cube))
    h2= Polyhedron(Matrix(vertices, number_type="fraction")).get_inequalities()
    hintersection = np.vstack((h1, h2))
    mat = Matrix(hintersection, number_type='fraction')
    mat.rep_type = RepType.INEQUALITY
    polyintersection = pcdd.Polyhedron(mat)
    vintersection = polyintersection.get_generators()
    if vintersection.row_size == 0 : break
  random_risk = steps[index-1]
```

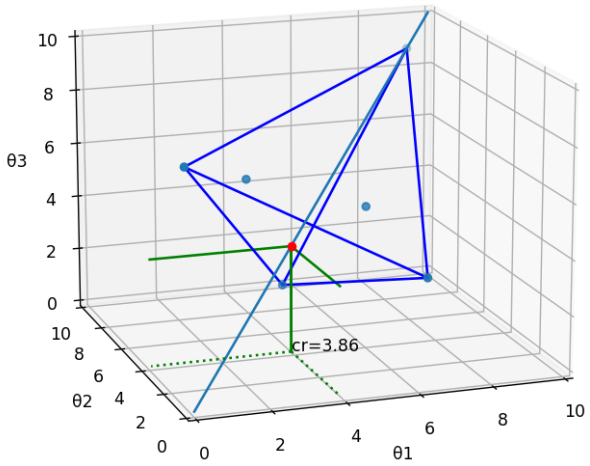

Рис. 2: Общий вид выпуклой оболочки, биссектрисы, точек пересечения и итогового рандомизированного риска $c_r^* = 3.86$

3 Маргинальное решение

Допустим, появляются наблюдения за перекрестком: $x = \{x_1, \ldots, x_q\}$. Они содержат информацию о состоянии светофора $\theta_j,\ j = 1, \ldots, k$.

Эта информация выражается в виде заданных условных вероятностей $P(x|\theta)$, $x \in X,\ \theta \in \Theta$.

Нам необходимо найти такую стратегию $s(\cdot) : X \to D$ — правило решения. Таких отображений может быть всего N^q штук. Каждому правилу s ставим в соответствие разбиение множества $X = D_1 + ... + D_N,\ D_j = \{x \in X : s(x) = d_j\},\ j = 1, \ldots, N$. Выражение для ожидаемого маргинального риска имеет вид:

$$L_i(s) = \sum_{t=1}^{N} l(\theta_i, d_t) * p_s(d_t|\theta_i)$$

В нашем условии задачи не даны вероятности $p_s(d_t|\theta_i)$, однако их можно вычислить:

$$p_s(d|\theta) = \sum_{x:\ s(x)=d} p(x|\theta).$$

Нам нужно сократить количество возможных действий, чтобы избежать долгого вычисления большого количества стратегий. В результате мы оставляем только определенные действия, которые могут быть выполнены.

1. Проехать перекресток, не сбрасывая скорость.
2. Остановиться, чтобы уступить дорогу, и продолжить движение.
3. Полностью остановиться.

Матрица потерь в таком случае примет вид:

Рис. 3: Матрица потерь

	x_1	x_2	x_3
θ_1	1	7	9
θ_2	4	2	9
θ_3	7	3	9

Добавим информацию о вероятности наблюдений при различных состояниях:

Рис. 4: Наблюдения

	x_1	x_2	x_3
θ_1	0.45	0.35	0.2
θ_2	0.25	0.5	0.25
θ_3	0.15	0.15	0.7

Найдем вероятности для разных стратегий.

```
def pr_count(s, i, t, strategy, P): #probability count function
  sum = 0
  for l in strategy[s][t]:
  sum += P[i, l-1]
  return sum
risks = np.array([[1,7,9],[4,2,9],[7,3,9]])
P = np.array([[0.45, 0.35, 0.2], [0.25, 0.5, 0.25], [0.15, 0.15, 0.7]])
strategy = []
create_strategy_combunation(strategy) #create list of stategy combination
ps = np.zeros((27, 3, 3))
for s in range(27):
  for i in range(3):
    for t in range(3):
      ps[s, i, t] = pr_count(s, i, t, strategy, P)
```

Рис. 5: Стратегии и вероятность попадания наблюдения в определенное разбиение

Стратегии	P_{11}	P_{21}	P_{31}	P_{12}	P_{22}	P_{32}	P_{13}	P_{23}	P_{33}
$x^{(1)} = \{x_1, x_2, x_3\} \cup \{\emptyset\} \cup \{\emptyset\}$	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
$x^{(2)} = \{x_1, x_2\} \cup \{x_3\} \cup \{\emptyset\}$	0.8	0.2	0.0	0.75	0.25	0.0	0.3	0.7	0.0
$x^{(3)} = \{x_1, x_2\} \cup \{\emptyset\} \cup \{x_3\}$	0.8	0.0	0.2	0.75	0.0	0.25	0.3	0.0	0.7
$x^{(4)} = \{x_1, x_3\} \cup \{x_2\} \cup \{\emptyset\}$	0.65	0.35	0.0	0.5	0.5	0.0	0.85	0.15	0.0
$x^{(5)} = \{x_1\} \cup \{x_2, x_3\} \cup \{\emptyset\}$	0.45	0.55	0.0	0.25	0.75	0.0	0.15	0.85	0.0
$x^{(6)} = \{x_1\} \cup \{x_2\} \cup \{x_3\}$	0.45	0.35	0.2	0.25	0.5	0.25	0.15	0.15	0.7
$x^{(7)} = \{x_1, x_3\} \cup \{\emptyset\} \cup \{x_2\}$	0.65	0.0	0.35	0.5	0.0	0.5	0.85	0.0	0.15
$x^{(8)} = \{x_1\} \cup \{x_3\} \cup \{x_2\}$	0.45	0.2	0.35	0.25	0.25	0.5	0.15	0.7	0.15
$x^{(9)} = \{x_1\} \cup \{\emptyset\} \cup \{x_2, x_3\}$	0.45	0.0	0.55	0.25	0.0	0.75	0.15	0.0	0.85
$x^{(10)} = \{x_2, x_3\} \cup \{x_1\} \cup \{\emptyset\}$	0.55	0.45	0.0	0.75	0.25	0.0	0.85	0.15	0.0
$x^{(11)} = \{x_2\} \cup \{x_1, x_3\} \cup \{\emptyset\}$	0.35	0.65	0.0	0.5	0.5	0.0	0.15	0.85	0.0
$x^{(12)} = \{x_2\} \cup \{x_1\} \cup \{x_3\}$	0.35	0.45	0.2	0.5	0.25	0.25	0.15	0.15	0.7
$x^{(13)} = \{x_3\} \cup \{x_1, x_2\} \cup \{\emptyset\}$	0.2	0.8	0.0	0.25	0.75	0.0	0.7	0.3	0.0
$x^{(14)} = \{\emptyset\} \cup \{x_1, x_2, x_3\} \cup \{\emptyset\}$	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
$x^{(15)} = \{\emptyset\} \cup \{x_1, x_2\} \cup \{x_3\}$	0.0	0.8	0.2	0.0	0.75	0.25	0.0	0.3	0.7
$x^{(16)} = \{x_3\} \cup \{x_1\} \cup \{x_2\}$	0.2	0.45	0.35	0.25	0.25	0.5	0.7	0.15	0.15
$x^{(17)} = \{\emptyset\} \cup \{x_1, x_3\} \cup \{x_2\}$	0.0	0.65	0.35	0.0	0.5	0.5	0.0	0.85	0.15
$x^{(18)} = \{\emptyset\} \cup \{x_1\} \cup \{x_2, x_3\}$	0.0	0.45	0.55	0.0	0.25	0.75	0.0	0.15	0.85
$x^{(19)} = \{x_2, x_3\} \cup \{\emptyset\} \cup \{x_1\}$	0.55	0.0	0.45	0.75	0.0	0.25	0.85	0.0	0.15
$x^{(20)} = \{x_2\} \cup \{x_3\} \cup \{x_1\}$	0.35	0.2	0.45	0.5	0.25	0.25	0.15	0.7	0.15
$x^{(21)} = \{x_2\} \cup \{\emptyset\} \cup \{x_1, x_3\}$	0.35	0.0	0.65	0.5	0.0	0.5	0.15	0.0	0.85
$x^{(22)} = \{x_3\} \cup \{x_2\} \cup \{x_1\}$	0.2	0.35	0.45	0.25	0.5	0.25	0.7	0.15	0.15
$x^{(23)} = \{\emptyset\} \cup \{x_2, x_3\} \cup \{x_1\}$	0.0	0.55	0.45	0.0	0.75	0.25	0.0	0.85	0.15
$x^{(24)} = \{\emptyset\} \cup \{x_2\} \cup \{x_1, x_3\}$	0.0	0.35	0.65	0.0	0.5	0.5	0.0	0.15	0.85
$x^{(25)} = \{x_3\} \cup \{\emptyset\} \cup \{x_1, x_2\}$	0.2	0.0	0.8	0.25	0.0	0.75	0.7	0.0	0.3
$x^{(26)} = \{\emptyset\} \cup \{x_3\} \cup \{x_1, x_2\}$	0.0	0.2	0.8	0.0	0.25	0.75	0.0	0.7	0.3
$x^{(27)} = \{\emptyset\} \cup \{\emptyset\} \cup \{x_1, x_2, x_3\}$	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

Найдем маргинальные риски $L_i(s)$.

```
def get_mar(i, s, Risks, ps):
  sum = 0
  for t in range(3):
    sum += Risks[i, t]*ps[s, i, t]
  return sum

L_mar = np.zeros((3, 27))
for i in range(3):
  for s in range(27):
    L_mar[i, s] = get_mar(i, s, risks, ps)

temp = []
for s in range(27):
  temp.append(np.amax(L_mar[:, s]))
min_max_marj = np.min(temp)
```

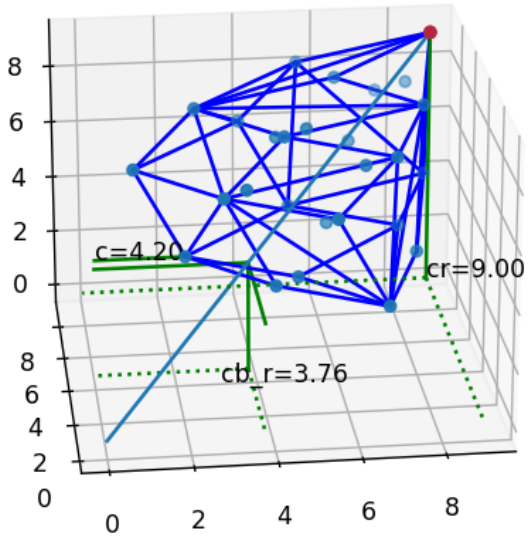

Рис. 6: Маргинальные риски, минмакс риск, выпуклая оболочка биссектриса, точки пересечения и итоговый рандомизированный риск

Рандомизированный маргинальный риск найдем похожим образом: построим выпуклую оболочку и найдем первую точку пересечения биссектрисы.
В данном случае сработал только метод куба, биссектриса оболочку не пересекает.
Получим оптимальный минимаксный маргниальный риск $c^* = 4.20$ для стратегии x_2 и рандомизированный маргниальный риск $c_r^* = 3.76$

4 Байесовское правило решения

В этом случае, в постановке задачи добавляется априорная информация $r_i \in R$ — вероятности состояний θ_i: $r_i = p(\theta_i), \sum_i P(r_i) = 1, r_i \leq \theta$.
Зададим априорные вероятности о состояних:

$$r_1 = P(\theta_1) = 0.5, \qquad r_2 = P(\theta_2) = 0.2, \qquad r_3 = P(\theta_3) = 0.3.$$

Байесовский риск теперь превращается в математическое ожидание маргинального риска по априорному распределению:

$$L(s) = \sum_{i=1}^{27} L_i(s) \cdot p(\theta_i).$$

Остается только из списка значений выбрать наименьшее. Это и будет байесовское правило принятия решений.

```
L_bs = []
for i in range(27):
  sum = 0
  for j in range(3):
    sum += L_mar[j, i]*prob[j]
L_bs.append(sum)

baer_risk = L_bs[np.argmin(L_bs)]
index = np.argmin(L_bs)
print(f"Bae's c* = {baer_risk} for strategy x{index+1}")
```

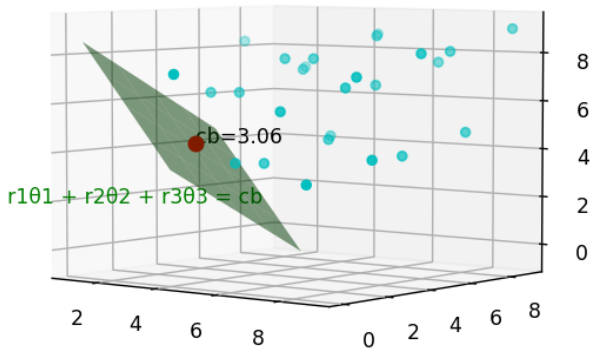

Рис. 7: Оптимальное решение по Баесу

То есть по Байесу $s^* = 2,\ c^*_{Bayes} = 3.06$. Как и полагается: $c^*_{Bayes} \leq c^*_r$.

5 Байесово действие

В данном случае мы рассматриваем байесовскую постановку задачи, но теперь у нас нет никаких наблюдений. Таким образом, мы должны искать байесово действие. Для этого мы вычисляем маргинальный риск следующим образом:

$$L(d_j) = \sum_{i=1}^{3} l(\theta_i, d_j) p(\theta_i)$$

Для решения необходимо:

$$L(d_j) \sim \min_{1 \leq j \leq N}$$

При любом распределении $p(\theta_1), p(\theta_2), p(\theta_1)$.

Решим эту задачу графически.

```
px, py = 0.5, 0.2
line = LineString([[px,py,0],[px,py,10]])
intersections = []

for i in range(len(L_mar)):
  model_x_data = np.linspace(0, 1, 51)
  model_y_data = np.linspace(0, 1, 51)
  X, Y = np.meshgrid(model_x_data, model_y_data)
  for k in range(len(Y)): # px + py <=1
    for j in range(len(Y[0])):
      if len(Y[0])-k<j : Y[k,j] = Y[k-1,j]
  Z = function(np.array([X, Y]),L_mar[i])
  x_dots = [2,2,10]
  y_dots = [2,8,2]
  x = X[0][x_dots]
  y = Y.T[0][y_dots]
  dots = [np.array([x,y,z]) for x,y,z in zip(x,y,function(np.array([x,y]),L_mar[i]))]
  intersect = intersect3D_SegmentPlane(line, dots)
  intersections.append(intersect[2])

  ax.plot_surface(X, Y, Z, alpha=0.3)
```

Получается при выборе $p_{\theta_1}, p_{\theta_2} = 0.5, 0.2$ Баесово действие - x_2

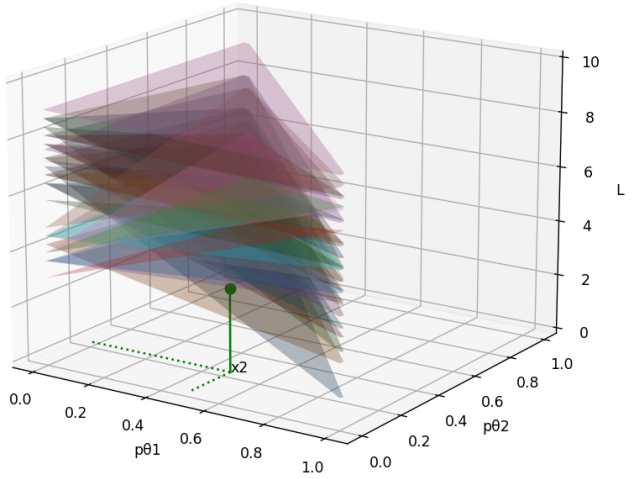

Рис. 8: Баесовское действие для трехмерной задачи

Наименее благоприятное распределение, при котором Баесовский риск равен рандомизированному маргинальному, найдем перебором значений вероятностей.

```
  def task5(L_mar, random_risk):
    X = np.linspace(0,1,1000)
    for x in X:
      Y =  np.linspace(0,1-x,1000)
      for y in Y:
        z = 1-x-y
        prob = np.array([x,y,z])
        baer_risk, _ = find_bier(prob, L_mar)
        if abs(baer_risk-random_risk)<0.00001: #epsilon
          return prob, baer_risk
```

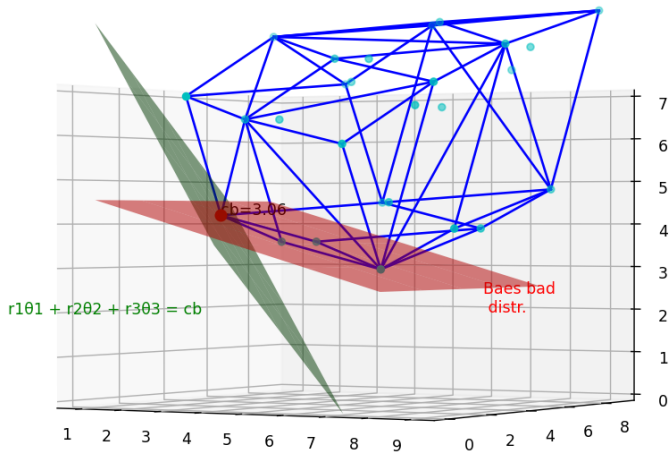

Рис. 9: Наименее благоприятное распределение для априорных вероятностей состояния природы

Баесовский риск $c^*_{Bayes} = c^*_r = 3.76$ для наименее благоприятного распределения $r_i = P(\theta_i) = \{0.22, 0, 0.78\}$.