exercise21

February 6, 2021

1 Exercise 21

Exercise 21. Provide a full rank design matrix Z for the pricing problem specified by the following risk class specification (assuming a multiplicative tariff structure).

	21-30y	31-40y	41-50y	51-60y
passenger car	2000	1800	1500	1600
delivery van	2200	1600	1400	1400
truck	2500	2000	1700	1600

Calculate a tariff using the different tariffication methods introduced above. ■

Wir wollen die einfachen Methoden ausprobieren. Dazu betrachten wir die zwei Covariaten: den Fahrzeugtyp und die Altersgruppe (in dieser Reihenfolge, d.h. $I = 3$ und $J = 4$). Die jeweiligen Schadenshöhen S_{ij} können der Aufgabe entnommen werden. Da nichts anderes angegeben ist, nehmen wir $v_{ij} = 1$ für alle $i \in [I]$ und alle $j \in [J]$ an. Wie in der Vorlesung sei $\mathbb{E}[X_{ij}] = v_{ij}\mu\chi_{1i}\chi_{2j}$. Um am Ende eine eindeutige Lösung zu erzwingen, setzen wir weiter $\mu = 1 = \chi_{11}$. Es verbleiben also χ_{ij} für $i = 2, 3$ und $j = 1, 2, 3, 4$ zubestimmen.

1.1 Methode von Bailey und Simons

Gemäß der Methode von Bailey und Simons gilt es

$$X^2 = \sum_{i=1}^{I}\sum_{j=1}^{J}\frac{(S_{ij} - v_{ij}\mu\chi_{1i}\chi_{2j})^2}{v_{ij}\mu\chi_{1i}\chi_{2j}} = \sum_{i=1}^{I}\sum_{j=1}^{J}\frac{(S_{ij} - \chi_{1i}\chi_{2j})^2}{\chi_{1i}\chi_{2j}}$$

zu minimieren. Wir nutzen die notwendige Optimalitätsbedingung erster Ordnung, um einen Minimierer zu finden. Es gilt

$$
\begin{aligned}
\frac{\partial}{\partial \chi_{1i}} X^2 &= \sum_{j=1}^{J} \frac{-2\left(S_{ij}-\chi_{1i}\chi_{2j}\right)\chi_{1i}\chi_{2j}^2-\left(S_{ij}-\chi_{1i}\chi_{2j}\right)^2\chi_{2j}}{\left(\chi_{1i}\chi_{2j}\right)^2} \\
&= \sum_{j=1}^{J} \frac{-2\left(S_{ij}-\chi_{1i}\chi_{2j}\right)\chi_{1i}\chi_{2j}-\left(S_{ij}-\chi_{1i}\chi_{2j}\right)^2}{\chi_{1i}^2\chi_{2j}} \\
&= \sum_{j=1}^{J} \frac{-2S_{ij}\chi_{1i}\chi_{2j}+2\chi_{1i}^2\chi_{2j}^2-S_{ij}^2+2S_{ij}\chi_{1i}\chi_{2j}-\chi_{1i}^2\chi_{2j}^2}{\chi_{1i}^2\chi_{2j}} \\
&= \sum_{j=1}^{J} \frac{\chi_{1i}^2\chi_{2j}^2-S_{ij}^2}{\chi_{1i}^2\chi_{2j}} \\
&= \sum_{j=1}^{J} \chi_{2j} - \frac{1}{\chi_{1i}^2}\sum_{j=1}^{J}\frac{S_{ij}^2}{\chi_{2j}}
\end{aligned}
$$

und somit erhalten wir durch $\frac{\partial}{\partial \chi_{1i}} X^2 = 0$ einen Schätzer für χ_{1i} für $i \in \{1,2\}$ und analog Schätzer für χ_{2j} für alle $j \in \{1,\dots,4\}$:

$$
\hat{\chi}_{1i} = \sqrt{\frac{\sum_{j=1}^{J} \frac{S_{ij}^2}{\chi_{2j}}}{\sum_{j=1}^{J} \chi_{2j}}} \quad \text{und} \quad \hat{\chi}_{2j} = \sqrt{\frac{\sum_{j=1}^{J} \frac{S_{ij}^2}{\chi_{1i}}}{\sum_{j=1}^{J} \chi_{1i}}}.
$$

Wir wollen hier einmal ausprobieren dieses System numerisch zu lösen und nutzen dafür Python und den Löser `fsolve}` aus dem `scipy`-Package:

[1]:
```
import numpy as np
from scipy.optimize import fsolve
```

[2]:
```
S = np.array([[2000, 1800, 1500, 1600],
              [2200, 1600, 1400, 1400],
              [2500, 2000, 1700, 1600]])
mu = 1
```

[3]:
```
# objective function
def objective(chi):
    chi1 = chi[:3]
    chi2 = chi[3:]

    res1 = np.sum(chi2) - 1/chi1**2 * np.sum(S**2/chi2,axis=1)
    res1[0] = chi1[0] - 1 # enforce chi11 = 1
    res2 = np.sum(chi1) - 1/chi2**2 * np.sum(S.transpose()**2/chi1,axis=1)

    return np.concatenate((res1, res2))
```

[4]:
```
# inital values for fsolve algorithm
chi1_init = np.ones(3)
chi2_init = 1000 * np.ones(4)
start = np.concatenate((chi1_init, chi2_init))
```

[5]:
```
def split(x,i=3):
    return x[:i], x[i:]
```

[6]:
```
# solve root-finding by fsolve
root = fsolve(objective, start)

chi1, chi2 = split(root)

print("The risk factors are given by \n chi1 = ", chi1.round(2), "\n chi2 = ",␣
↪chi2.round(2))
```

```
The risk factors are given by
 chi1 =  [1.   0.96 1.13]
 chi2 =  [2175.59 1751.38 1491.35 1493.27]
```

[7]:
```
def calc_tariff(mu, chi1, chi2):
    tariff = mu * chi1.reshape((-1,1)) @ chi2.reshape((1,-1))
    print("Considering a multiplikative tariff structure we get \n", np.
↪round(tariff,0))
    return tariff
```

[8]:
```
# tariff structure
tariff_bs = calc_tariff(mu, chi1, chi2)
```

```
Considering a multiplikative tariff structure we get
 [[2176. 1751. 1491. 1493.]
 [2079. 1674. 1425. 1427.]
 [2456. 1977. 1684. 1686.]]
```

1.2 Bailey-Jung Methode

Als nächstes probieren wir die Methode von Bailey und Jung. Dazu betrachten wir die Spalten- bzw. Zeilensummen

$$\sum_{j=1}^{J} v_{ij}\mu\chi_{1i}\chi_{2j} = \sum_{j=1}^{J} S_{ij}$$
$$\sum_{i=1}^{I} v_{ij}\mu\chi_{1i}\chi_{2j} = \sum_{i=1}^{I} S_{ij}.$$

Mit der Setzung $\mu = 1$ und $v_{ij} = 1$ vereinfachen sich diese zu

$$\sum_{j=1}^{J} \chi_{1i}\chi_{2j} = \sum_{j=1}^{J} S_{ij} =: S_{\text{row}}$$
$$\sum_{i=1}^{I} \chi_{1i}\chi_{2j} = \sum_{i=1}^{I} S_{ij} =: S_{\text{col}}.$$

Setzen wir erneut $\chi_{11} = 1$ und lösen diese Gleichungen numerisch.

[9]:
```
def margsums(chi):
    chi1 = chi[:3]
    chi2 = chi[3:]

    S_row = np.sum(S, axis=1)
    S_col = np.sum(S, axis=0)

    res1 = chi1 * np.sum(chi2) - S_row
    res2 = chi2 * np.sum(chi1) - S_col

    return np.concatenate((res1, res2))

# inital values for fsolve algorithm
chi1_init = np.ones(3)
chi2_init = 1000 * np.ones(4)
start = np.concatenate((chi1_init, chi2_init))

# solve root-finding by fsolve
root_bj = fsolve(margsums, start)

chi1_bj, chi2_bj = split(root)

print("The risk factors are given by \n chi1 = ", chi1.round(2), "\n chi2 = ",␣
 ↪chi2.round(2))
```

```
The risk factors are given by
 chi1 =  [1.   0.96 1.13]
 chi2 =  [2175.59 1751.38 1491.35 1493.27]
```

[10]:
```
tariff_bailey_jung = calc_tariff(mu, chi1_bj, chi2_bj)
```

```
Considering a multiplikative tariff structure we get
 [[2176. 1751. 1491. 1493.]
 [2079. 1674. 1425. 1427.]
 [2456. 1977. 1684. 1686.]]
```

1.3 Log-Normal-Approximation

Abschließend wollen wir noch die log-Normal-Approximation testen. Dazu definieren wir wie in der Vorlesung $X_{ij} = \log\left(\frac{S_{ij}}{v_{ij}}\right) = \log(S_{ij}) \sim \mathcal{N}(\beta_0 + \beta_{1i} + \beta_{2j}, \sigma^2)$. Wie üblich sei $M = I \cdot J$ und $m := m(i,j) := (i-1)J + j$.

[11]:
```
I,J = np.shape(S)
M = I * J
m = lambda i,j: (i-1) * J + j
```

Wir lesen die Daten aus S neu ein, um mit dem `pandas`-Paket automatisch die Langform der Kontingenztabelle zu erhalten.

[12]:
```
import pandas as pd
```

[13]:
```
S_data = pd.DataFrame({'type': ['passenger car', 'delivery van', 'truck'],␣
↪'21-30y': [2000,2200,2500], '31-40y': [1800, 1600, 2000], '41-50y': [1500,␣
↪1400, 1700], '51-60y': [1600, 1400, 1600]})
S_data
```

[13]:

	type	21-30y	31-40y	41-50y	51-60y
0	passenger car	2000	1800	1500	1600
1	delivery van	2200	1600	1400	1400
2	truck	2500	2000	1700	1600

[14]:
```
Z_twiddle_df = S_data.melt(id_vars="type")
Z_twiddle_df
```

[14]:

	type	variable	value
0	passenger car	21-30y	2000
1	delivery van	21-30y	2200
2	truck	21-30y	2500
3	passenger car	31-40y	1800
4	delivery van	31-40y	1600
5	truck	31-40y	2000
6	passenger car	41-50y	1500
7	delivery van	41-50y	1400
8	truck	41-50y	1700
9	passenger car	51-60y	1600
10	delivery van	51-60y	1400
11	truck	51-60y	1600

[15]:
```
Z_twiddle = Z_twiddle_df.to_numpy()
Z_twiddle
```

[15]:
```
array([['passenger car', '21-30y', 2000],
       ['delivery van', '21-30y', 2200],
       ['truck', '21-30y', 2500],
```

```
       ['passenger car', '31-40y', 1800],
       ['delivery van', '31-40y', 1600],
       ['truck', '31-40y', 2000],
       ['passenger car', '41-50y', 1500],
       ['delivery van', '41-50y', 1400],
       ['truck', '41-50y', 1700],
       ['passenger car', '51-60y', 1600],
       ['delivery van', '51-60y', 1400],
       ['truck', '51-60y', 1600]], dtype=object)
```

Wir konvertieren nun diese Tabelle in die aus der Vorlesung bekannte Langform und erhalten eine Matrix $\tilde{Z}$. Außerdem erhalten wir den Vektor $\mathbf{X} = S_{ij}$.

[16]:
```
X = np.log(Z_twiddle[:,-1].astype(float))
```

[17]:
```
veh_type = Z_twiddle[:, 0]
veh_type_bin = np.zeros((M,3))
vehicles = ['passenger car', 'delivery van', 'truck']
for vehicle in vehicles:
    veh_type_bin[:,vehicles.index(vehicle)][veh_type == vehicle] = 1
```

[18]:
```
age_group = Z_twiddle[:, 1]
age_group_bin = np.zeros((M,4))
ages = ['21-30y', '31-40y', '41-50y', '51-60y']
for age in ages:
    age_group_bin[:,ages.index(age)][age_group == age] = 1
```

[19]:
```
Z_twiddle_bin = np.concatenate((veh_type_bin, age_group_bin), axis=1)
print("Ztwiddle = \n", Z_twiddle_bin)
```

```
Ztwiddle =
 [[1. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0.]
 [1. 0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 1.]]
```

Die Matrix $\tilde{Z}$ hat nicht vollen Rang.

[20]:
```
np.linalg.matrix_rank(Z_twiddle_bin)
```

[20]: 6

Wir setzen $\beta_{11} = \beta_{21} = 0$. Dazu fügen wir eine 1-Spalte zu Beginn ein und streichen jeweils die erste Spalte jedes Kovariats. Dies liefert uns die Matrix Z.

[21]:
```
Z = Z_twiddle_bin.copy()
Z[:,0] = np.ones(M)
Z = np.delete(Z,I,1)
print("Z = \n", Z)
```

```
Z =
 [[1. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 0.]
 [1. 0. 1. 0. 0. 0.]
 [1. 0. 0. 1. 0. 0.]
 [1. 1. 0. 1. 0. 0.]
 [1. 0. 1. 1. 0. 0.]
 [1. 0. 0. 0. 1. 0.]
 [1. 1. 0. 0. 1. 0.]
 [1. 0. 1. 0. 1. 0.]
 [1. 0. 0. 0. 0. 1.]
 [1. 1. 0. 0. 0. 1.]
 [1. 0. 1. 0. 0. 1.]]
```

Die Matrix Z hat nun vollen Rang.

[22]:
```
np.linalg.matrix_rank(Z)
```

[22]: 6

Damit können wir nun den Maximum-Likelihood-Estimator $\hat{\beta}$ angeben. Gemäß Vorlesung ist Maximum-Likelihood-Estimator $\hat{\beta}^{\mathrm{MLE}}$ gegeben durch

$$\hat{\beta}^{\mathrm{MLE}} = \left(Z^\top Z\right)^{-1} Z^\top \mathbf{X}.$$

Man beachte, dass die Einträge $\beta_{11} = \beta_1$ und $\beta_{21} = \beta_5$ durch deren Setzung auf Null nicht enthalten sind.

[23]:
```
beta = np.linalg.inv(Z.T @ Z) @ Z.T @ X
print(np.round(beta,2))
```

```
[ 7.69 -0.06  0.11 -0.22 -0.38 -0.37]
```

Somit erhalten wir die Schätzer

$$\begin{array}{llll} \beta_0 = 7.69 & & & \\ \beta_{11} = 0 & \beta_{12} = -0.06 & \beta_{13} = 0.11 & \\ \beta_{21} = 0 & \beta_{22} = -0.22 & \beta_{23} = -0.38 & \beta_{24} = -0.37 \end{array}$$

[24]:
```
beta = np.insert(beta, 1, 0)
beta = np.insert(beta, I+1, 0)
print(beta)
```

```
[ 7.68799786  0.          -0.05624928  0.1134168   0.          -0.2156526
 -0.37510989 -0.37380526]
```

Daraus können wir nun wieder die Risikofaktoren χ berechnen, nämlich durch $\chi_{ij} = e^{\beta_{ij}}$ und $\mu = \beta_0$.

[25]:
```
mu = np.exp(beta)[0]
chi1_lognormal = np.exp(beta)[1:I+1]
chi2_lognormal = np.exp(beta)[I:]

print("mu = ", np.round(mu,2), "\nchi1 = ", np.round(chi1_lognormal,2), "\nchi2␣
 ↪= ", np.round(chi2_lognormal,2))
```

```
mu =  2182.0
chi1 =  [1.   0.95 1.12]
chi2 =  [1.12 1.   0.81 0.69 0.69]
```

Berechnen wir damit die Tarife, so erhalten wir folgendes Resultat:

[26]:
```
tariff_lognormal = calc_tariff(mu, chi1_lognormal, chi2_lognormal)
```

```
Considering a multiplikative tariff structure we get
 [[2444. 2182. 1759. 1500. 1501.]
 [2310. 2063. 1663. 1417. 1419.]
 [2738. 2444. 1970. 1680. 1682.]]
```