Pythagorean Fuzzy TODIM Ranking

July 1, 2021

[1]:
```
import pfn                          # for fuzzy types
import math                         # for sqrt and other functions
import numpy as np                  # for linear algebra
import pandas as pd                 # for tabular output
from scipy.stats import rankdata # for ranking the candidates
```

1 Step 0 - Obtaining and preprocessing data

[2]:
```
attributes_data = pd.read_csv('../data/criteria.csv')
attributes_data
```

[2]:

	Criteria	Name	Rank	Ideally
0	Price	Price	3	Lower
1	Quality	Quality	5	Higher
2	EC	Energy Consumption	2	Lower
3	GD	Green Design	1	Higher
4	DS	Delivery Speed	6	Higher
5	CSR	Corporate Social Responsibility	4	Higher
6	EE	Employee Education	7	Higher

[3]:
```
benefit_attributes = set()
attributes = []
rankings = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Criteria'])
    rankings.append(row['Rank'])
    n += 1

    if row['Ideally'] == 'Higher':
        benefit_attributes.add(i)
```

[4]:
```
pd.DataFrame(attributes, columns=['Criteria Name'])
```

[4]:

	Criteria Name
0	Price

```
1       Quality
2            EC
3            GD
4            DS
5           CSR
6            EE
```

[5]:
```
rankings = np.array(rankings)
weights = 2 * (n + 1 - rankings) / (n * (n + 1))

pd.DataFrame(weights, columns=['Weights'])
```

[5]:
```
    Weights
0  0.178571
1  0.107143
2  0.214286
3  0.250000
4  0.071429
5  0.142857
6  0.035714
```

[6]:
```
original_dataframe = pd.read_csv('../data/alternatives (fuzzy).csv')
candidates = list(original_dataframe['Name'])
raw_data = pd.DataFrame(original_dataframe, columns=attributes).values.tolist()

m = len(raw_data)
n = len(raw_data[0])

pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[6]:
```
           Price     Quality            EC         GD            DS  \
S1    (.085,.01)   (.68,.08)  (.6375,.075)    (.8,.1)  (.2125,.025)
S2  (.2125,.025)   (.17,.02)      (.85,.1)  (.64,.08)  (.2125,.025)
S3      (.85,.1)  (.085,.01)  (.6375,.075)  (.51,.06)  (.2125,.025)
S4    (.085,.01)    (.85,.1)  (.6375,.075)   (.85,.1)       (.8,.1)
S5    (.425,.05)   (.34,.04)      (.85,.1)  (.51,.06)  (.2125,.025)
S6    (.065,.01)  (.51,.061)  (.6375,.075)  (.68,.08)      (.85,.1)

           CSR           EE
S1    (.7,.08)      (.85,.1)
S2   (.75,.08)  (.6375,.075)
S3  (.65,.076)  (.6375,.075)
S4    (.85,.1)      (.85,.1)
S5   (.75,.08)  (.6375,.075)
S6    (.85,.1)      (.85,.1)
```

[7]:
```
raw_data = [[pfn.parse_pfn(item) for item in row] for row in raw_data]
pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[7]:

	Price	Quality	EC	GD	\
S1	(0.085, 0.01)	(0.68, 0.08)	(0.6375, 0.075)	(0.8, 0.1)	
S2	(0.2125, 0.025)	(0.17, 0.02)	(0.85, 0.1)	(0.64, 0.08)	
S3	(0.85, 0.1)	(0.085, 0.01)	(0.6375, 0.075)	(0.51, 0.06)	
S4	(0.085, 0.01)	(0.85, 0.1)	(0.6375, 0.075)	(0.85, 0.1)	
S5	(0.425, 0.05)	(0.34, 0.04)	(0.85, 0.1)	(0.51, 0.06)	
S6	(0.065, 0.01)	(0.51, 0.061)	(0.6375, 0.075)	(0.68, 0.08)	

	DS	CSR	EE
S1	(0.2125, 0.025)	(0.7, 0.08)	(0.85, 0.1)
S2	(0.2125, 0.025)	(0.75, 0.08)	(0.6375, 0.075)
S3	(0.2125, 0.025)	(0.65, 0.076)	(0.6375, 0.075)
S4	(0.8, 0.1)	(0.85, 0.1)	(0.85, 0.1)
S5	(0.2125, 0.025)	(0.75, 0.08)	(0.6375, 0.075)
S6	(0.85, 0.1)	(0.85, 0.1)	(0.85, 0.1)

2 Step 1 - Normalizing the ratings and weights

[8]:
```
for j in range(n):
    if j not in benefit_attributes:
        for i in range(m):
            raw_data[i][j] = -raw_data[i][j]

pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[8]:

	Price	Quality	EC	GD	\
S1	(0.01, 0.085)	(0.68, 0.08)	(0.075, 0.6375)	(0.8, 0.1)	
S2	(0.025, 0.2125)	(0.17, 0.02)	(0.1, 0.85)	(0.64, 0.08)	
S3	(0.1, 0.85)	(0.085, 0.01)	(0.075, 0.6375)	(0.51, 0.06)	
S4	(0.01, 0.085)	(0.85, 0.1)	(0.075, 0.6375)	(0.85, 0.1)	
S5	(0.05, 0.425)	(0.34, 0.04)	(0.1, 0.85)	(0.51, 0.06)	
S6	(0.01, 0.065)	(0.51, 0.061)	(0.075, 0.6375)	(0.68, 0.08)	

	DS	CSR	EE
S1	(0.2125, 0.025)	(0.7, 0.08)	(0.85, 0.1)
S2	(0.2125, 0.025)	(0.75, 0.08)	(0.6375, 0.075)
S3	(0.2125, 0.025)	(0.65, 0.076)	(0.6375, 0.075)
S4	(0.8, 0.1)	(0.85, 0.1)	(0.85, 0.1)
S5	(0.2125, 0.025)	(0.75, 0.08)	(0.6375, 0.075)
S6	(0.85, 0.1)	(0.85, 0.1)	(0.85, 0.1)

[9]:
```
max_weight = max(weights)
for i in range(n):
```

```
    weights[i] /= max_weight

pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[9]:

	Weight
Price	0.714286
Quality	0.428571
EC	0.857143
GD	1.000000
DS	0.285714
CSR	0.571429
EE	0.142857

3 Step 2 - Calculating Dominance Degrees

[10]:
```
# The loss attenuation factor
theta = 2.5
```

[11]:
```
phi = np.zeros((n, m, m))

weight_sum = sum(weights)

for c in range(n):
    for i in range(m):
        for j in range(m):
            pic = raw_data[i][c]
            pjc = raw_data[j][c]
            val = 0
            if pic > pjc:
                val = math.sqrt((pic.distance(pjc)) * weights[c] / weight_sum)
            if pic < pjc:
                val = -1.0 / theta * math.sqrt(weight_sum * pic.distance(pjc) /␣
↪weights[c])
            phi[c][i][j] = val
```

[12]:
```
delta = np.zeros((m, m))
for i in range(m):
    for j in range(m):
        delta[i,j] = sum(phi[:,i,j])

pd.DataFrame(data=delta, index=candidates, columns=candidates)
```

[12]:

	S1	S2	S3	S4	S5	S6
S1	0.000000	0.622632	1.094783	-2.526386	0.762338	-1.440690
S2	-2.957800	0.000000	0.245667	-4.921384	-0.010759	-4.309130
S3	-3.595614	-1.408338	0.000000	-5.321140	-1.238177	-4.794761

```
S4  0.701042  1.363427  1.498947  0.000000  1.500441 -0.002496
S5 -3.192777 -0.563682  0.074355 -5.163070  0.000000 -4.569714
S6 -0.463511  1.099182  1.285355 -1.143332  1.269214  0.000000
```

4 Step 3 - Calculate ratings from the normalised dominance degree values

[13]:
```
delta_sums = np.zeros(m)
for i in range(m):
    delta_sums[i] = sum(delta[i,:])
pd.DataFrame(data=delta_sums,index=candidates,columns=['Sum'])
```

[13]:
```
           Sum
S1  -1.487323
S2 -11.953406
S3 -16.358030
S4   5.061361
S5 -13.414887
S6   2.046908
```

[14]:
```
delta_min = min(delta_sums)
delta_max = max(delta_sums)
pd.DataFrame(data=[delta_min, delta_max], columns=['Value'], index=['Minimum',␣
 ↪'Maximum'])
```

[14]:
```
              Value
Minimum -16.358030
Maximum   5.061361
```

[15]:
```
ratings = (delta_sums - delta_min) / (delta_max - delta_min)
pd.DataFrame(data=ratings, index=candidates, columns=['Rating'])
```

[15]:
```
      Rating
S1  0.694264
S2  0.205637
S3  0.000000
S4  1.000000
S5  0.137406
S6  0.859265
```

5 Step 4 - Create rankings based on calculated ξ_i values

[16]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
```

```
    return storage[::-1]
```

[17]:
```
result = rank_according_to(ratings)
pd.DataFrame(data=result, index=range(1, m + 1), columns=['Name'])
```

[17]:

	Name
1	S4
2	S6
3	S1
4	S2
5	S5
6	S3

[18]:
```
print("The best candidate/alternative according to C* is " + str(result[0]))
print("The preferences in descending order are " + ", ".join(str(r) for r in␣
 ↪result) + ".")
```

```
The best candidate/alternative according to C* is S4
The preferences in descending order are S4, S6, S1, S2, S5, S3.
```