CPT TODIM Ranking

March 18, 2022

1 CPT TODIM Ranking

[1]:
```
import math                       # For sqrt and other stuff
import numpy as np                # For linear algebra
import pandas as pd               # For tabular output
from scipy.stats import rankdata # For ranking the candidates based on score
```

2 Step 0 - Obtaining and preprocessing the data

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

[2]:

	Indicator	Name	Unit	Ideally	Rank
0	C1	The average wage	US Dollar	Higher	11
1	C2	The employment rate	% of the working age population	Higher	7
2	C3	Income inequality	ratio	Lower	14
3	C4	Labor force	Thousand persons	Higher	1
4	C5	Poverty gap	Ratio	Lower	10
5	C6	Poverty rate	Ratio	Lower	9
6	C7	Working hours	Hours/worker	Higher	8
7	C8	Women in politics	Percentage		
8	C9	Population density	Ratio		
9	C10	Adult education level	% of 25-64 year-old		
10	C11	Spending on tertiary education	% of education spending		
11	C12	International student mobility	% of students enrolled		
12	C13	Tertiary graduation rate	% of the same level		
13	C14	Social spending	% of GDP		

```
7   Higher     5
8    Lower     2
9   Higher     6
10  Higher     4
11  Higher     3
12  Higher    13
13  Higher    12
```

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

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

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

[4]:
```
rankings = np.array(rankings)
weights = 2 * (n + 1 - rankings) / (n * (n + 1))

pd.DataFrame(zip(attributes, weights), columns=['Attribute', 'Weight'])
```

[4]:

	Attribute	Weight
0	C1	0.038095
1	C2	0.076190
2	C3	0.009524
3	C4	0.133333
4	C5	0.047619
5	C6	0.057143
6	C7	0.066667
7	C8	0.095238
8	C9	0.123810
9	C10	0.085714
10	C11	0.104762
11	C12	0.114286
12	C13	0.019048
13	C14	0.028571

[5]:
```
print(f'The sum of the weights is {sum(weights):0.2f}')
```

```
The sum of the weights is 1.00
```

[6]:
```
original_dataframe = pd.read_csv('../data/alternatives.csv').T
```

```
updated_dataframe = original_dataframe.drop(original_dataframe.index[0])

candidates = np.array(updated_dataframe.index)
raw_data = updated_dataframe.to_numpy()

[m, n] = updated_dataframe.shape

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

[6]:

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	\
CA	53198.17	64.73	0.31	20199.55	0.3	0.12	1670.0	51.7	4.0	57.88	
FR	46480.62	66.02	0.29	29682.22	0.25	0.08	1505.0	52.9	122.0	36.89	
DE	53637.8	76.09	0.28	43769.63	0.25	0.1	1386.1	33.3	237.0	29.06	
IT	39189.37	59.07	0.33	25941.4	0.4	0.13	1717.8	27.8	205.0	19.32	
JP	38617.47	77.95	0.33	68863.34	0.33	0.15	1644.0	15.8	347.0	51.92	
UK	47226.09	75.61	0.35	33964.07	0.34	0.11	1538.0	30.8	275.0	45.78	
USA	65835.58	62.56	0.39	163538.7	0.38	0.17	1779.0	16.7	36.0	47.43	

	C11	C12	C13	C14
CA	49.052	12.917	54.4	20.89
FR	77.838	10.201	54.31	31.68
DE	82.723	8.373	49.33	24.76
IT	61.715	5.311	56.07	25.36
JP	32.416	4.265	36.87	23.51
UK	24.991	17.918	54.47	24.49
USA	35.205	5.18	55.41	30.02

3 Step 1 - Normalizing the Ratings and Weights

[7]:
```
for j in range(n):
    column = raw_data[:, j]
    if j in benefit_attributes:
        raw_data[:, j] /= sum(column)
    else:
        column = 1 / column
        raw_data[:, j] = column / sum(column)

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

[7]:

	C1	C2	C3	C4	C5	C6	C7	\
CA	0.154563	0.134286	0.148467	0.052336	0.148568	0.138507	0.148578	
FR	0.135045	0.136962	0.158707	0.076905	0.178282	0.20776	0.133898	
DE	0.15584	0.157853	0.164375	0.113405	0.178282	0.166208	0.12332	
IT	0.113861	0.122544	0.139469	0.067213	0.111426	0.127852	0.152831	
JP	0.1122	0.161712	0.139469	0.178421	0.135062	0.110805	0.146265	
UK	0.137211	0.156857	0.1315	0.087999	0.13109	0.151098	0.136834	

```
USA   0.19128  0.129784  0.118013   0.42372  0.117291  0.097769  0.158275

            C8        C9       C10       C11       C12       C13       C14
CA    0.225764  0.828939  0.200777   0.13478  0.201309  0.150751    0.1156
FR    0.231004  0.027178  0.127966  0.213876  0.158981  0.150502  0.175309
DE    0.145415  0.013991  0.100805  0.227298  0.130492  0.136701  0.137015
IT    0.121397  0.016174  0.067018  0.169575  0.082771  0.155379  0.140335
JP    0.068996  0.009555  0.180103   0.08907  0.066469  0.102173  0.130098
UK    0.134498  0.012057  0.158804  0.068668  0.279249  0.150945  0.135521
USA   0.072926  0.092104  0.164528  0.096733  0.080729   0.15355  0.166123
```

[8]:
```
max_weight = max(weights)
weights /= max_weight

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

[8]:
```
       Weight
C1   0.285714
C2   0.571429
C3   0.071429
C4   1.000000
C5   0.357143
C6   0.428571
C7   0.500000
C8   0.714286
C9   0.928571
C10  0.642857
C11  0.785714
C12  0.857143
C13  0.142857
C14  0.214286
```

4 Step 2 - Calculating Dominance Degrees

α and β are parameters of decision makers' risk attitude and they are viewed as preference degrees in the domain of gain and loss, respectively.

λ is the parameter of loss aversion that is more sensitive to loss than gain.

γ and δ are the parameters describing the curvature of the weighting function. They express the differences of diminishing sensitivity in the domain of gains and losses.

[9]:
```
alpha = 0.88
beta = 0.88
gamma = 0.61
delta = 0.69
lambda_ = 2.25
```

[10]:
```
inv_gamma = 1 / gamma
inv_delta = 1 / delta
```

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

for i in range(m):
    for k in range(m):
        for j in range(n):
            if raw_data[i, j] >= raw_data[k, j]:
                w_gamma = weights[j] ** gamma
                pi[i, k, j] = w_gamma / (
                    (w_gamma + (1 - weights[j]) ** gamma) ** inv_gamma
                )
            else:
                w_delta = weights[j] ** delta
                pi[i, k, j] = w_delta / (
                    (w_delta + (1 - weights[j]) ** delta) ** inv_delta
                )

        pi[i, k, :] /= max(pi[i, k, :])
```

[12]:
```
pi_sums = np.zeros((m, m))

for i in range(m):
    for k in range(m):
        pi_sums[i, k] = sum(pi[i, k, :])
```

[13]:
```
phi = np.zeros((n, m, m))

for i in range(m):
    for k in range(m):
        for j in range(n):
            x_ij = raw_data[i, j]
            x_kj = raw_data[k, j]
            val = 0.0
            if x_ij > x_kj:
                val = pi[i, k, j] * ((x_ij - x_kj) ** alpha) / pi_sums[i, k]
            if x_ij < x_kj:
                val = (
                    -lambda_
                    * pi_sums[i, k]
                    * ((x_kj - x_ij) ** beta)
                    / pi[i, k, j]
                )
            phi[j, i, k] = val
```

[14]:
```
big_phi = np.zeros((m, m))
for i in range(m):
    for j in range(m):
        big_phi[i, j] = sum(phi[:, i, j])
```

[15]:
```
big_phi_sums = np.zeros(m)
for i in range(m):
    big_phi_sums[i] = sum(big_phi[i, :])

pd.DataFrame(data=big_phi_sums,index=candidates,columns=['Sum'])
```

[15]:

	Sum
CA	-58.856412
FR	-55.494989
DE	-77.989684
IT	-131.312161
JP	-153.753302
UK	-98.104911
USA	-120.010459

[16]:
```
big_phi_min = min(big_phi_sums)
big_phi_max = max(big_phi_sums)

pd.DataFrame(data=[big_phi_min, big_phi_max], columns=['Value'],␣
↪index=['Minimum', 'Maximum'])
```

[16]:

	Value
Minimum	-153.753302
Maximum	-55.494989

[17]:
```
ratings = (big_phi_sums - big_phi_min) / (big_phi_max - big_phi_min)
pd.DataFrame(data=ratings, index=candidates, columns=['Rating'])
```

[17]:

	Rating
CA	0.965790
FR	1.000000
DE	0.771066
IT	0.228389
JP	0.000000
UK	0.566348
USA	0.343410

5 Step 4 - Create rankings based on calculated ξ_i values

[18]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[19]:
```
result = rank_according_to(ratings)
pd.DataFrame(data=result, index=range(1, m + 1), columns=['Name'])
```

[19]:

	Name
1	FR
2	CA
3	DE
4	UK
5	USA
6	IT
7	JP

[20]:
```
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 FR
The preferences in descending order are FR, CA, DE, UK, USA, IT, JP.
```