NR TOPSIS Ranking

August 4, 2021

1 TOPSIS Ranking

[1]:
```python
import numpy as np                   # for linear algebra
import pandas as pd                  # for tabular output
from scipy.stats import rankdata # for ranking the candidates
```

1.1 Step 0 - Obtaining and processing the data

The data from the Excel sheet is saved into CSV files and stored in the `data` folder at the root of the project. The criteria, their rankings, the players' scores based on the mentioned criteria are stored in Numpy arrays and processed for the next step.

Note that an attribute can be beneficial attribute (in which case, we will want to maximize it's contribution) or a cost attribute (which we will need to minimize). We call the set of beneficial attributes J_1 and that of cost attributes $J_2 = J_1^C$.

[2]:
```python
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = bowlers_data
```

[3]:
```python
attributes_data = pd.read_csv(data['weights'])
attributes_data
```

[3]:

	Name	Ranking	Ideally
0	SR	1	Lower
1	Econ	2	Lower
2	Avg	3	Lower
3	Wkts	4	Higher
4	Runs	5	Lower
5	Inns	6	Higher
6	TBB	7	Higher
7	4w	8	Higher

```
8    Mat         9  Higher
```

[4]:
```
benefit_attributes = set()
attributes = []
ranks = []
n = 0

for i, row in attributes_data.iterrows():
    attributes.append(row['Name'])
    ranks.append(float(row['Ranking']))
    n += 1

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

ranks = np.array(ranks)
```

[5]:
```
weights = 2 * (n + 1 - ranks) / (n * (n + 1))
pd.DataFrame(data=weights, index=attributes, columns=['Weight'])
```

[5]:

	Weight
SR	0.200000
Econ	0.177778
Avg	0.155556
Wkts	0.133333
Runs	0.111111
Inns	0.088889
TBB	0.066667
4w	0.044444
Mat	0.022222

[6]:
```
original_dataframe = pd.read_csv(data['scores'])
candidates = original_dataframe['Name'].to_numpy()
raw_data = pd.DataFrame(original_dataframe, columns=attributes).to_numpy()

dimensions = raw_data.shape
m = dimensions[0]
n = dimensions[1]

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

[6]:

	SR	Econ	Avg	Wkts	Runs	Inns	TBB	4w	Mat
Andre Russell	16.45	9.51	26.09	11.0	287.0	12.0	181.0	0.0	14.0
Ben Stokes	16.83	11.23	31.50	6.0	189.0	6.0	101.0	0.0	9.0
Chris Morris	15.23	9.27	23.54	13.0	306.0	9.0	198.0	0.0	9.0
Dwayne Bravo	22.45	8.02	30.00	11.0	330.0	12.0	247.0	0.0	12.0
Imran Tahir	14.85	6.70	16.58	26.0	431.0	17.0	386.0	2.0	17.0

Jofra Archer	23.45	6.77	26.45	11.0	291.0	11.0	258.0	0.0	11.0
Kagiso Rabada	11.28	7.83	14.72	25.0	368.0	12.0	282.0	2.0	12.0
Keemo Paul	18.11	8.72	26.33	9.0	237.0	8.0	163.0	0.0	8.0
Lasith Malinga	16.81	9.77	27.38	16.0	438.0	12.0	269.0	2.0	12.0
Moeen Ali	25.00	6.76	28.17	6.0	169.0	9.0	150.0	0.0	11.0
Mohammad Nabi	21.88	6.65	24.25	8.0	194.0	8.0	175.0	1.0	8.0
Rashid Khan	21.18	6.28	22.18	17.0	377.0	15.0	360.0	0.0	15.0
Sam Curran	19.80	9.79	32.30	10.0	323.0	9.0	198.0	1.0	9.0
Sunil Narine	26.60	7.83	34.70	10.0	347.0	12.0	266.0	0.0	12.0
Trent Boult	22.80	8.58	32.60	5.0	163.0	5.0	114.0	0.0	5.0

1.2 Step 1 - Normalizing the ratings

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[7]:
```
for j in range(n):
    column = raw_data[:,j]
    min_val = np.min(column)
    max_val = np.max(column)
    denom = max_val - min_val
    if j in benefit_attributes:
        raw_data[:,j] = (raw_data[:,j] - min_val) / denom
    else:
        raw_data[:,j] = (max_val - raw_data[:,j]) / denom

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

[7]:

	SR	Econ	Avg	Wkts	Runs	Inns \
Andre Russell	0.662533	0.347475	0.430931	0.285714	0.549091	0.583333
Ben Stokes	0.637728	0.000000	0.160160	0.047619	0.905455	0.083333
Chris Morris	0.742167	0.395960	0.558559	0.380952	0.480000	0.333333
Dwayne Bravo	0.270888	0.648485	0.235235	0.285714	0.392727	0.583333
Imran Tahir	0.766971	0.915152	0.906907	1.000000	0.025455	1.000000
Jofra Archer	0.205614	0.901010	0.412913	0.285714	0.534545	0.500000
Kagiso Rabada	1.000000	0.686869	1.000000	0.952381	0.254545	0.583333
Keemo Paul	0.554178	0.507071	0.418919	0.190476	0.730909	0.250000
Lasith Malinga	0.639034	0.294949	0.366366	0.523810	0.000000	0.583333
Moeen Ali	0.104439	0.903030	0.326827	0.047619	0.978182	0.333333
Mohammad Nabi	0.308094	0.925253	0.523023	0.142857	0.887273	0.250000
Rashid Khan	0.353786	1.000000	0.626627	0.571429	0.221818	0.833333
Sam Curran	0.443864	0.290909	0.120120	0.238095	0.418182	0.333333
Sunil Narine	0.000000	0.686869	0.000000	0.238095	0.330909	0.583333
Trent Boult	0.248042	0.535354	0.105105	0.000000	1.000000	0.000000

TBB 4w Mat

```
Andre Russell    0.280702  0.0  0.750000
Ben Stokes       0.000000  0.0  0.333333
Chris Morris     0.340351  0.0  0.333333
Dwayne Bravo     0.512281  0.0  0.583333
Imran Tahir      1.000000  1.0  1.000000
Jofra Archer     0.550877  0.0  0.500000
Kagiso Rabada    0.635088  1.0  0.583333
Keemo Paul       0.217544  0.0  0.250000
Lasith Malinga   0.589474  1.0  0.583333
Moeen Ali        0.171930  0.0  0.500000
Mohammad Nabi    0.259649  0.5  0.250000
Rashid Khan      0.908772  0.0  0.833333
Sam Curran       0.340351  0.5  0.333333
Sunil Narine     0.578947  0.0  0.583333
Trent Boult      0.045614  0.0  0.000000
```

1.3 Step 2 - Calculating the Weighted Normalized Ratings

$$v_{ij} = w_j r_{ij}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

[8]:
```
raw_data *= weights
pd.DataFrame(data=raw_data, index=candidates, columns=attributes)
```

[8]:
```
                       SR      Econ       Avg      Wkts      Runs      Inns  \
Andre Russell    0.132507  0.061773  0.067034  0.038095  0.061010  0.051852
Ben Stokes       0.127546  0.000000  0.024914  0.006349  0.100606  0.007407
Chris Morris     0.148433  0.070393  0.086887  0.050794  0.053333  0.029630
Dwayne Bravo     0.054178  0.115286  0.036592  0.038095  0.043636  0.051852
Imran Tahir      0.153394  0.162694  0.141074  0.133333  0.002828  0.088889
Jofra Archer     0.041123  0.160180  0.064231  0.038095  0.059394  0.044444
Kagiso Rabada    0.200000  0.122110  0.155556  0.126984  0.028283  0.051852
Keemo Paul       0.110836  0.090146  0.065165  0.025397  0.081212  0.022222
Lasith Malinga   0.127807  0.052435  0.056990  0.069841  0.000000  0.051852
Moeen Ali        0.020888  0.160539  0.050840  0.006349  0.108687  0.029630
Mohammad Nabi    0.061619  0.164489  0.081359  0.019048  0.098586  0.022222
Rashid Khan      0.070757  0.177778  0.097475  0.076190  0.024646  0.074074
Sam Curran       0.088773  0.051717  0.018685  0.031746  0.046465  0.029630
Sunil Narine     0.000000  0.122110  0.000000  0.031746  0.036768  0.051852
Trent Boult      0.049608  0.095174  0.016350  0.000000  0.111111  0.000000

                      TBB        4w       Mat
Andre Russell    0.018713  0.000000  0.016667
Ben Stokes       0.000000  0.000000  0.007407
Chris Morris     0.022690  0.000000  0.007407
Dwayne Bravo     0.034152  0.000000  0.012963
Imran Tahir      0.066667  0.044444  0.022222
```

Jofra Archer	0.036725	0.000000	0.011111
Kagiso Rabada	0.042339	0.044444	0.012963
Keemo Paul	0.014503	0.000000	0.005556
Lasith Malinga	0.039298	0.044444	0.012963
Moeen Ali	0.011462	0.000000	0.011111
Mohammad Nabi	0.017310	0.022222	0.005556
Rashid Khan	0.060585	0.000000	0.018519
Sam Curran	0.022690	0.022222	0.007407
Sunil Narine	0.038596	0.000000	0.012963
Trent Boult	0.003041	0.000000	0.000000

1.4 Step 3 - Identifying PIS (A^*) and NIS (A^-)

$$A^* = \{v_1^*, v_2^*, \dots, v_n^*\}$$
$$A^- = \{v_1^-, v_2^-, \dots, v_n^-\}$$

And we define

$$v_j^* = \max(v_{ij}), \ \text{if} j \in J_1$$
$$v_j^* = \min(v_{ij}), \ \text{if} j \in J_2$$
$$v_j^- = \min(v_{ij}), \ \text{if} j \in J_1$$
$$v_j^- = \max(v_{ij}), \ \text{if} j \in J_2$$

where $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$.

[9]:
```
a_pos = np.copy(weights)
a_neg = np.zeros(n)

pd.DataFrame(data=[a_pos, a_neg], index=["$A^*$", "$A^-$"], columns=attributes)
```

[9]:
```
        SR      Econ       Avg      Wkts      Runs      Inns       TBB  \
$A^*$  0.2  0.177778  0.155556  0.133333  0.111111  0.088889  0.066667
$A^-$  0.0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

             4w       Mat
$A^*$  0.044444  0.022222
$A^-$  0.000000  0.000000
```

1.5 Step 4 and 5 - Calculating Separation Measures and Similarities to PIS

The separation or distance between the alternatives can be measured by the n-dimensional Euclidean distance. The separation from the PIS A^* and NIS A^- are S^* and S^- respectively.

$$S_i^* = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^*\right)^2}$$

$$S_i^- = \sqrt{\sum_{j=1}^{n} \left(v_{ij} - v_j^-\right)^2}$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

We also calculate

$$C_i^* = \frac{S_i^-}{S_i^* + S_i^-}, \text{ where } i = 1, 2, \ldots, m$$

[10]:
```
sp = np.zeros(m)
sn = np.zeros(m)
cs = np.zeros(m)

for i in range(m):
    diff_pos = raw_data[i] - a_pos
    diff_neg = raw_data[i] - a_neg
    sp[i] = np.sqrt(diff_pos @ diff_pos)
    sn[i] = np.sqrt(diff_neg @ diff_neg)
    cs[i] = sn[i] / (sp[i] + sn[i])

pd.DataFrame(data=zip(sp, sn, cs), index=candidates, columns=["$S^*$", "$S^-$",␣
↪"$C^*$"])
```

[10]:

	\$S^*\$	\$S^-\$	\$C^*\$
Andre Russell	0.207621	0.185358	0.471674
Ben Stokes	0.288852	0.164804	0.363279
Chris Morris	0.191566	0.203492	0.515094
Dwayne Bravo	0.239656	0.157935	0.397230
Imran Tahir	0.119727	0.320159	0.727823
Jofra Archer	0.224961	0.199751	0.470322
Kagiso Rabada	0.109768	0.320403	0.744827
Keemo Paul	0.213923	0.180645	0.457831
Lasith Malinga	0.221858	0.183265	0.452369
Moeen Ali	0.261022	0.204400	0.439172
Mohammad Nabi	0.213779	0.221073	0.508387
Rashid Khan	0.181835	0.249050	0.577996
Sam Curran	0.260104	0.126541	0.327279
Sunil Narine	0.295574	0.147027	0.332188
Trent Boult	0.284633	0.155375	0.353119

1.6 Step 6 - Ranking the candidates/alternatives

We choose the candidate with the maximum C^* or rank all the alternatives in descending order according to their C^* values. This process can also be done for the S^* and S^- values.

[11]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[12]:
```
cs_order = rank_according_to(cs)
sp_order = rank_according_to(sp)
sn_order = rank_according_to(sn)

pd.DataFrame(data=zip(cs_order, sp_order[::-1], sn_order), index=range(1, m +␣
 ↪1),
             columns=["$C^*$", "$S^*$", "$S^-$"])
```

[12]:

	C^*	S^*	S^-
1	Kagiso Rabada	Kagiso Rabada	Kagiso Rabada
2	Imran Tahir	Imran Tahir	Imran Tahir
3	Rashid Khan	Rashid Khan	Rashid Khan
4	Chris Morris	Chris Morris	Mohammad Nabi
5	Mohammad Nabi	Andre Russell	Moeen Ali
6	Andre Russell	Mohammad Nabi	Chris Morris
7	Jofra Archer	Keemo Paul	Jofra Archer
8	Keemo Paul	Lasith Malinga	Andre Russell
9	Lasith Malinga	Jofra Archer	Lasith Malinga
10	Moeen Ali	Dwayne Bravo	Keemo Paul
11	Dwayne Bravo	Sam Curran	Ben Stokes
12	Ben Stokes	Moeen Ali	Dwayne Bravo
13	Trent Boult	Trent Boult	Trent Boult
14	Sunil Narine	Ben Stokes	Sunil Narine
15	Sam Curran	Sunil Narine	Sam Curran

[13]:
```
print("The best candidate/alternative according to C* is " + cs_order[0])
print("The preferences in descending order are " + ", ".join(cs_order) + ".")
```

```
The best candidate/alternative according to C* is Kagiso Rabada
The preferences in descending order are Kagiso Rabada, Imran Tahir, Rashid Khan,
Chris Morris, Mohammad Nabi, Andre Russell, Jofra Archer, Keemo Paul, Lasith
Malinga, Moeen Ali, Dwayne Bravo, Ben Stokes, Trent Boult, Sunil Narine, Sam
Curran.
```