NR TOPSIS Ranking

August 4, 2021

1 TOPSIS Ranking

[1]:
```
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]:
```
bowlers_data = {
    'weights': '../data/bowling_criteria.csv',
    'scores': '../data/bowlers.csv',
}
batsmen_data = {
    'weights': '../data/batting_criteria.csv',
    'scores': '../data/batsmen.csv',
}
data = batsmen_data
```

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

[3]:

	Name	Ranking	Ideally
0	SR	1	Higher
1	Avg	2	Higher
2	Runs	3	Higher
3	Inn	4	Higher
4	NO	5	Higher
5	6s	6	Higher
6	4s	7	Higher
7	100s	8	Higher

```
8    50s        9  Higher
9    Mat       10  Higher
10    HS       11  Higher
11    BF       12  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    0.153846
Avg   0.141026
Runs  0.128205
Inn   0.115385
NO    0.102564
6s    0.089744
4s    0.076923
100s  0.064103
50s   0.051282
Mat   0.038462
HS    0.025641
BF    0.012821
```

[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	Avg	Runs	Inn	NO	6s	4s	100s	50s	Mat	HS	BF
AB de Villiers	154.00	44.20	442.0	13.0	3.0	26.0	31.0	0.0	5.0	13.0	82.0	287.0
Andre Russel	204.81	56.67	510.0	13.0	4.0	52.0	31.0	0.0	4.0	14.0	80.0	249.0
Ben Stokes	124.24	20.50	123.0	9.0	3.0	4.0	8.0	0.0	0.0	9.0	46.0	99.0
Chris Gayle	153.60	40.83	490.0	13.0	1.0	34.0	45.0	0.0	4.0	13.0	99.0	319.0
Chris Lynn	139.65	31.15	405.0	13.0	0.0	22.0	41.0	0.0	4.0	13.0	82.0	290.0
David Warner	143.86	69.20	692.0	12.0	2.0	21.0	57.0	1.0	8.0	12.0	100.0	481.0
Faf Du Plessis	123.36	36.00	396.0	12.0	1.0	15.0	36.0	0.0	3.0	12.0	96.0	321.0
Jonny Bairstow	157.24	55.63	445.0	10.0	2.0	18.0	48.0	1.0	2.0	10.0	114.0	283.0
Jos Buttler	151.70	38.88	311.0	8.0	0.0	14.0	38.0	0.0	3.0	8.0	89.0	205.0
Kane Williamson	120.00	22.29	156.0	9.0	2.0	5.0	12.0	0.0	1.0	9.0	70.0	130.0
Kieron Pollard	156.74	34.88	279.0	14.0	6.0	22.0	14.0	0.0	1.0	16.0	83.0	178.0
Marcus Stoinis	135.25	52.75	211.0	10.0	6.0	10.0	14.0	0.0	0.0	10.0	46.0	156.0
Moeen Ali	165.41	27.50	220.0	10.0	2.0	17.0	16.0	0.0	2.0	11.0	66.0	133.0
Quinton de Kock	132.91	35.27	529.0	16.0	1.0	25.0	45.0	0.0	4.0	16.0	81.0	398.0
Shane Watson	127.56	23.41	398.0	17.0	0.0	20.0	42.0	0.0	3.0	17.0	96.0	312.0
Steve Smith	116.00	39.88	319.0	10.0	2.0	4.0	30.0	0.0	3.0	12.0	73.0	275.0

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	Avg	Runs	Inn	NO	6s	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.427880	0.486653	0.560633	0.555556	0.500000	0.458333	0.469388	0.0	0.625	0.555556	0.529412	0.492147
Andre Russel	1.000000	0.742710	0.680141	0.555556	0.666667	1.000000	0.469388	0.0	0.500	0.666667	0.500000	0.392670
Ben Stokes	0.092782	0.000000	0.000000	0.111111	0.500000	0.000000	0.000000	0.0	0.000	0.111111	0.000000	0.000000
Chris Gayle	0.423376	0.417454	0.644991	0.555556	0.166667	0.625000	0.755102	0.0	0.500	0.555556	0.779412	0.575916
Chris Lynn	0.266299	0.218686	0.495606	0.555556	0.000000	0.375000	0.673469	0.0	0.500	0.555556	0.529412	0.500000
David Warner	0.313703	1.000000	1.000000	0.444444	0.333333	0.354167	1.000000	1.0	1.000	0.444444	0.794118	1.000000
Faf Du Plessis	0.082874	0.318275	0.479789	0.444444	0.166667	0.229167	0.571429	0.0	0.375	0.444444	0.735294	0.581152
Jonny Bairstow	0.464362	0.721355	0.565905	0.222222	0.333333	0.291667	0.816327	1.0	0.250	0.222222	1.000000	0.481675
Jos Buttler	0.401982	0.377413	0.330404	0.000000	0.000000	0.208333	0.612245	0.0	0.375	0.000000	0.632353	0.277487
Kane Williamson	0.045040	0.036756	0.057996	0.111111	0.333333	0.020833	0.081633	0.0	0.125	0.111111	0.352941	0.081152
Kieron Pollard	0.458732	0.295277	0.274165	0.666667	1.000000	0.375000	0.122449	0.0	0.125	0.888889	0.544118	0.206806
Marcus Stoinis	0.216755	0.662218	0.154657	0.222222	1.000000	0.125000	0.122449	0.0	0.000	0.222222	0.000000	0.149215
Moeen Ali	0.556356	0.143737	0.170475	0.222222	0.333333	0.270833	0.163265	0.0	0.250	0.333333	0.294118	0.089005
Quinton de Kock	0.190406	0.303285	0.713533	0.888889	0.166667	0.437500	0.755102	0.0	0.500	0.888889	0.514706	0.782723
Shane Watson	0.130166	0.059754	0.483304	1.000000	0.000000	0.333333	0.693878	0.0	0.375	1.000000	0.735294	0.557592
Steve Smith	0.000000	0.397947	0.344464	0.222222	0.333333	0.000000	0.448980	0.0	0.375	0.444444	0.397059	0.460733

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	Avg	Runs	Inn	NO	6s
AB de Villiers	0.065828	0.068631	0.071876	0.064103	0.051282	0.041132
Andre Russel	0.153846	0.104741	0.087198	0.064103	0.068376	0.089744
Ben Stokes	0.014274	0.000000	0.000000	0.012821	0.051282	0.000000
Chris Gayle	0.065135	0.058872	0.082691	0.064103	0.017094	0.056090
Chris Lynn	0.040969	0.030840	0.063539	0.064103	0.000000	0.033654
David Warner	0.048262	0.141026	0.128205	0.051282	0.034188	0.031784
Faf Du Plessis	0.012750	0.044885	0.061511	0.051282	0.017094	0.020566
Jonny Bairstow	0.071440	0.101730	0.072552	0.025641	0.034188	0.026175
Jos Buttler	0.061843	0.053225	0.042360	0.000000	0.000000	0.018697
Kane Williamson	0.006929	0.005183	0.007435	0.012821	0.034188	0.001870
Kieron Pollard	0.070574	0.041642	0.035149	0.076923	0.102564	0.033654
Marcus Stoinis	0.033347	0.093390	0.019828	0.025641	0.102564	0.011218
Moeen Ali	0.085593	0.020271	0.021856	0.025641	0.034188	0.024306
Quinton de Kock	0.029293	0.042771	0.091479	0.102564	0.017094	0.039263
Shane Watson	0.020025	0.008427	0.061962	0.115385	0.000000	0.029915
Steve Smith	0.000000	0.056121	0.044162	0.025641	0.034188	0.000000

	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.036107	0.000000	0.032051	0.021368	0.013575	0.006310
Andre Russel	0.036107	0.000000	0.025641	0.025641	0.012821	0.005034
Ben Stokes	0.000000	0.000000	0.000000	0.004274	0.000000	0.000000
Chris Gayle	0.058085	0.000000	0.025641	0.021368	0.019985	0.007384
Chris Lynn	0.051805	0.000000	0.025641	0.021368	0.013575	0.006410
David Warner	0.076923	0.064103	0.051282	0.017094	0.020362	0.012821
Faf Du Plessis	0.043956	0.000000	0.019231	0.017094	0.018854	0.007451
Jonny Bairstow	0.062794	0.064103	0.012821	0.008547	0.025641	0.006175
Jos Buttler	0.047096	0.000000	0.019231	0.000000	0.016214	0.003558
Kane Williamson	0.006279	0.000000	0.006410	0.004274	0.009050	0.001040
Kieron Pollard	0.009419	0.000000	0.006410	0.034188	0.013952	0.002651
Marcus Stoinis	0.009419	0.000000	0.000000	0.008547	0.000000	0.001913
Moeen Ali	0.012559	0.000000	0.012821	0.012821	0.007541	0.001141
Quinton de Kock	0.058085	0.000000	0.025641	0.034188	0.013198	0.010035
Shane Watson	0.053375	0.000000	0.019231	0.038462	0.018854	0.007149
Steve Smith	0.034537	0.000000	0.019231	0.017094	0.010181	0.005907

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

$$A^* = \{w_1, w_2, \ldots, w_n\}$$

$$A^- = \{0, 0, \ldots, 0\}$$

[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	Avg	Runs	Inn	NO	6s	4s	\
A^*	0.153846	0.141026	0.128205	0.115385	0.102564	0.089744	0.076923	
A^-	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

	100s	50s	Mat	HS	BF
A^*	0.064103	0.051282	0.038462	0.025641	0.012821
A^-	0.000000	0.000000	0.000000	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^*
AB de Villiers	0.174382	0.160163	0.478748
Andre Russel	0.116700	0.248776	0.680691
Ben Stokes	0.302746	0.054920	0.153552

```
Chris Gayle       0.182300  0.164425  0.474222
Chris Lynn        0.226352  0.126221  0.357999
David Warner      0.154195  0.238726  0.607567
Faf Du Plessis    0.237810  0.110955  0.318137
Jonny Bairstow    0.175557  0.179480  0.505525
Jos Buttler       0.245161  0.108004  0.305817
Kane Williamson   0.300392  0.040603  0.119072
Kieron Pollard    0.202568  0.164377  0.447960
Marcus Stoinis    0.236055  0.147292  0.384226
Moeen Ali         0.241691  0.105719  0.304306
Quinton de Kock   0.204539  0.169953  0.453822
Shane Watson      0.244431  0.153702  0.386057
Steve Smith       0.257682  0.094443  0.268208
```

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	Andre Russel	Andre Russel	Andre Russel
2	David Warner	David Warner	David Warner
3	Jonny Bairstow	AB de Villiers	Jonny Bairstow
4	AB de Villiers	Jonny Bairstow	Quinton de Kock
5	Chris Gayle	Chris Gayle	Chris Gayle
6	Quinton de Kock	Kieron Pollard	Kieron Pollard
7	Kieron Pollard	Quinton de Kock	AB de Villiers
8	Shane Watson	Chris Lynn	Shane Watson
9	Marcus Stoinis	Marcus Stoinis	Marcus Stoinis
10	Chris Lynn	Faf Du Plessis	Chris Lynn
11	Faf Du Plessis	Moeen Ali	Faf Du Plessis
12	Jos Buttler	Shane Watson	Jos Buttler
13	Moeen Ali	Jos Buttler	Moeen Ali

```
14      Steve Smith      Steve Smith      Steve Smith
15       Ben Stokes  Kane Williamson       Ben Stokes
16  Kane Williamson       Ben Stokes  Kane Williamson
```

[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 Andre Russel
The preferences in descending order are Andre Russel, David Warner, Jonny
Bairstow, AB de Villiers, Chris Gayle, Quinton de Kock, Kieron Pollard, Shane
Watson, Marcus Stoinis, Chris Lynn, Faf Du Plessis, Jos Buttler, Moeen Ali,
Steve Smith, Ben Stokes, Kane Williamson.
```