TODIM Ranking

May 28, 2021

1 TODIM Ranking

[1]:
```
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
```

1.1 Step 0 - Obtaining and preprocessing data

[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
AB de Villiers	154.00	44.20	442.0	13.0	3.0	26.0	31.0	0.0	5.0	13.0
Andre Russel	204.81	56.67	510.0	13.0	4.0	52.0	31.0	0.0	4.0	14.0
Ben Stokes	124.24	20.50	123.0	9.0	3.0	4.0	8.0	0.0	0.0	9.0
Chris Gayle	153.60	40.83	490.0	13.0	1.0	34.0	45.0	0.0	4.0	13.0

Chris Lynn	139.65	31.15	405.0	13.0	0.0	22.0	41.0	0.0	4.0	13.0
David Warner	143.86	69.20	692.0	12.0	2.0	21.0	57.0	1.0	8.0	12.0
Faf Du Plessis	123.36	36.00	396.0	12.0	1.0	15.0	36.0	0.0	3.0	12.0
Jonny Bairstow	157.24	55.63	445.0	10.0	2.0	18.0	48.0	1.0	2.0	10.0
Jos Buttler	151.70	38.88	311.0	8.0	0.0	14.0	38.0	0.0	3.0	8.0
Kane Williamson	120.00	22.29	156.0	9.0	2.0	5.0	12.0	0.0	1.0	9.0
Kieron Pollard	156.74	34.88	279.0	14.0	6.0	22.0	14.0	0.0	1.0	16.0
Marcus Stoinis	135.25	52.75	211.0	10.0	6.0	10.0	14.0	0.0	0.0	10.0
Moeen Ali	165.41	27.50	220.0	10.0	2.0	17.0	16.0	0.0	2.0	11.0
Quinton de Kock	132.91	35.27	529.0	16.0	1.0	25.0	45.0	0.0	4.0	16.0
Shane Watson	127.56	23.41	398.0	17.0	0.0	20.0	42.0	0.0	3.0	17.0
Steve Smith	116.00	39.88	319.0	10.0	2.0	4.0	30.0	0.0	3.0	12.0

	HS	BF
AB de Villiers	82.0	287.0
Andre Russel	80.0	249.0
Ben Stokes	46.0	99.0
Chris Gayle	99.0	319.0
Chris Lynn	82.0	290.0
David Warner	100.0	481.0
Faf Du Plessis	96.0	321.0
Jonny Bairstow	114.0	283.0
Jos Buttler	89.0	205.0
Kane Williamson	70.0	130.0
Kieron Pollard	83.0	178.0
Marcus Stoinis	46.0	156.0
Moeen Ali	66.0	133.0
Quinton de Kock	81.0	398.0
Shane Watson	96.0	312.0
Steve Smith	73.0	275.0

1.2 Step 1 - Normalizing the Ratings And Weights

$$P_{ij} = \begin{cases} \frac{x_{ij}}{\sum_{k=1}^{m} x_{kj}} & \text{if } j \in J_1 \\ \frac{\frac{1}{x_{ij}}}{\sum_{k=1}^{m} \frac{1}{x_{kj}}} & \text{if } j \in J_2 \end{cases}$$

$$w_{rc} = \frac{w_c}{w_r}$$

and $w_r = \max\{w_c | c = 1, 2, \ldots, n\}$

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

[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]:

	SR	Avg	Runs	Inn	NO	6s
AB de Villiers	0.066773	0.070266	0.074587	0.068783	0.085714	0.084142
Andre Russel	0.088803	0.090090	0.086061	0.068783	0.114286	0.168285
Ben Stokes	0.053869	0.032589	0.020756	0.047619	0.085714	0.012945
Chris Gayle	0.066599	0.064908	0.082686	0.068783	0.028571	0.110032
Chris Lynn	0.060551	0.049520	0.068343	0.068783	0.000000	0.071197
David Warner	0.062376	0.110009	0.116774	0.063492	0.057143	0.067961
Faf Du Plessis	0.053488	0.057230	0.066824	0.063492	0.028571	0.048544
Jonny Bairstow	0.068178	0.088436	0.075093	0.052910	0.057143	0.058252
Jos Buttler	0.065775	0.061808	0.052481	0.042328	0.000000	0.045307
Kane Williamson	0.052031	0.035435	0.026325	0.047619	0.057143	0.016181
Kieron Pollard	0.067961	0.055450	0.047081	0.074074	0.171429	0.071197
Marcus Stoinis	0.058643	0.083858	0.035606	0.052910	0.171429	0.032362
Moeen Ali	0.071720	0.043717	0.037125	0.052910	0.057143	0.055016
Quinton de Kock	0.057628	0.056070	0.089268	0.084656	0.028571	0.080906
Shane Watson	0.055309	0.037215	0.067162	0.089947	0.000000	0.064725
Steve Smith	0.050296	0.063398	0.053831	0.052910	0.057143	0.012945

	4s	100s	50s	Mat	HS	BF
AB de Villiers	0.061024	0.0	0.106383	0.066667	0.062932	0.069728
Andre Russel	0.061024	0.0	0.085106	0.071795	0.061397	0.060496
Ben Stokes	0.015748	0.0	0.000000	0.046154	0.035303	0.024052
Chris Gayle	0.088583	0.0	0.085106	0.066667	0.075979	0.077502
Chris Lynn	0.080709	0.0	0.085106	0.066667	0.062932	0.070457
David Warner	0.112205	0.5	0.170213	0.061538	0.076746	0.116861
Faf Du Plessis	0.070866	0.0	0.063830	0.061538	0.073676	0.077988
Jonny Bairstow	0.094488	0.5	0.042553	0.051282	0.087490	0.068756
Jos Buttler	0.074803	0.0	0.063830	0.041026	0.068304	0.049806
Kane Williamson	0.023622	0.0	0.021277	0.046154	0.053722	0.031584
Kieron Pollard	0.027559	0.0	0.021277	0.082051	0.063699	0.043246
Marcus Stoinis	0.027559	0.0	0.000000	0.051282	0.035303	0.037901
Moeen Ali	0.031496	0.0	0.042553	0.056410	0.050652	0.032313
Quinton de Kock	0.088583	0.0	0.085106	0.082051	0.062164	0.096696
Shane Watson	0.082677	0.0	0.063830	0.087179	0.073676	0.075802
Steve Smith	0.059055	0.0	0.063830	0.061538	0.056025	0.066812

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

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

[8]:

	Weight
SR	1.000000
Avg	0.916667
Runs	0.833333
Inn	0.750000
NO	0.666667
6s	0.583333
4s	0.500000
100s	0.416667
50s	0.333333
Mat	0.250000
HS	0.166667
BF	0.083333

1.3 Step 2 - Calculating Dominance Degrees

For the contribution of each criteria, we have:

$$\Phi_c\left(A_i, A_j\right) = \begin{cases} \sqrt{\frac{(P_{ic}-P_{jc})w_{rc}}{\sum_{c=1}^{n} w_{rc}}} & \text{if } P_{ic} - P_{jc} > 0 \\ 0 & \text{if } P_{ic} - P_{jc} = 0 \\ -\frac{1}{\theta}\sqrt{\frac{\left(\sum_{c=1}^{n} w_{rc}\right)(P_{jc}-P_{ic})}{w_{rc}}} & \text{if } P_{ic} - P_{jc} < 0 \end{cases}$$

Combining all contributions, we get the dominance degrees:

$$\delta\left(A_i, A_j\right) = \sum_{c=1}^{n} \Phi_c\left(A_i, A_j\right)$$

Here $c = 1, 2, \ldots, n$, $i, j = 1, 2, \ldots, m$.

[9]:
```
# The loss attenuation factor
theta = 1.0
```

[10]:
```
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 - pjc) * weights[c] / weight_sum)
            if pic < pjc:
```

```
                val = -1.0 / theta * math.sqrt(weight_sum * (pjc - pic) /␣
↪weights[c])
            phi[c, i, j] = val
```

[11]:
```
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)
```

[11]:

	AB de Villiers	Andre Russel	Ben Stokes	Chris Gayle
AB de Villiers	0.000000	-2.863567	0.541653	-2.736826
Andre Russel	-1.432863	0.000000	0.680447	-2.185471
Ben Stokes	-8.636770	-9.667204	0.000000	-9.146171
Chris Gayle	-1.464281	-2.969557	-0.175313	0.000000
Chris Lynn	-2.701454	-3.741018	-0.417572	-3.823306
David Warner	-1.209961	-2.478412	0.286893	-0.915043
Faf Du Plessis	-3.656001	-4.732884	-0.356786	-3.705243
Jonny Bairstow	-3.131000	-4.366829	0.157937	-3.398367
Jos Buttler	-5.721169	-6.457348	-1.096931	-6.463471
Kane Williamson	-8.286399	-8.948843	-0.484118	-8.413792
Kieron Pollard	-4.391405	-5.261568	0.525750	-5.592233
Marcus Stoinis	-7.120193	-7.819790	0.373489	-7.790279
Moeen Ali	-7.067200	-8.058174	-0.172304	-7.215517
Quinton de Kock	-2.139380	-2.687265	-0.192804	-1.683666
Shane Watson	-3.144248	-4.090498	-0.432750	-3.807919
Steve Smith	-5.171623	-5.638931	-0.338728	-5.684441

	Chris Lynn	David Warner	Faf Du Plessis	Jonny Bairstow
AB de Villiers	-0.470107	-8.323082	-1.469548	-4.718239
Andre Russel	-1.227112	-8.352768	-1.751344	-4.861276
Ben Stokes	-8.182355	-13.270051	-7.802967	-10.830438
Chris Gayle	0.285547	-8.043642	0.118903	-4.553716
Chris Lynn	0.000000	-9.426578	-1.984101	-5.766008
David Warner	-0.167186	0.000000	0.639631	-0.486275
Faf Du Plessis	-2.293111	-9.522781	0.000000	-5.872369
Jonny Bairstow	-2.207302	-6.057709	-1.841390	0.000000
Jos Buttler	-4.292833	-11.468555	-4.064842	-8.271405
Kane Williamson	-7.306546	-12.686198	-7.081497	-10.064896
Kieron Pollard	-3.562238	-9.797233	-4.159653	-7.506049
Marcus Stoinis	-6.826716	-12.139600	-6.384429	-9.027501
Moeen Ali	-6.017898	-11.455425	-5.522270	-7.842208
Quinton de Kock	-0.034919	-8.317030	-0.465049	-5.084213
Shane Watson	-1.373950	-9.392831	-1.139287	-5.636553
Steve Smith	-4.242396	-10.580745	-3.466620	-6.795844

	Jos Buttler	Kane Williamson	Kieron Pollard	Marcus Stoinis	\
AB de Villiers	-0.478668	0.563287	-1.747245	-0.811863	
Andre Russel	-0.406260	0.668864	-1.363234	-0.227213	
Ben Stokes	-6.242437	-3.047484	-7.635036	-4.509206	
Chris Gayle	0.409858	0.014354	-1.793310	-1.094065	
Chris Lynn	-0.686636	-0.283309	-2.544914	-1.401158	
David Warner	0.574301	0.791753	-1.890787	-0.340003	
Faf Du Plessis	-0.466721	-0.110308	-2.814779	-1.472636	
Jonny Bairstow	-0.088623	0.652716	-2.297592	-0.518570	
Jos Buttler	0.000000	-0.965622	-3.323164	-2.234822	
Kane Williamson	-5.310323	0.000000	-6.188112	-3.995256	
Kieron Pollard	-2.951892	0.470782	0.000000	-0.161458	
Marcus Stoinis	-4.712187	-1.131119	-4.804666	0.000000	
Moeen Ali	-3.900915	-0.055282	-4.854388	-2.047408	
Quinton de Kock	-0.542137	-0.004426	-1.384756	-1.273152	
Shane Watson	-0.422441	-0.299606	-1.951912	-1.619388	
Steve Smith	-1.879711	0.002998	-3.729563	-1.917626	

	Moeen Ali	Quinton de Kock	Shane Watson	Steve Smith
AB de Villiers	0.263601	-3.177333	-2.701426	0.402402
Andre Russel	0.585822	-3.176525	-2.930901	-0.184972
Ben Stokes	-5.279731	-9.286885	-8.464639	-6.335553
Chris Gayle	-0.283686	-2.311074	-0.825192	-0.137100
Chris Lynn	-0.675130	-4.204480	-2.472064	-0.754139
David Warner	0.431890	-0.987474	-0.655316	0.684438
Faf Du Plessis	-0.849702	-4.643568	-2.159345	-0.519110
Jonny Bairstow	-0.049678	-4.253256	-2.720731	-0.668040
Jos Buttler	-1.985314	-6.235162	-4.644032	-2.984853
Kane Williamson	-3.472313	-8.463717	-7.646521	-5.268632
Kieron Pollard	-0.692945	-5.152669	-4.855014	-3.049238
Marcus Stoinis	-2.986853	-7.949185	-7.297779	-4.810067
Moeen Ali	0.000000	-7.394533	-6.366026	-4.321665
Quinton de Kock	-0.407927	0.000000	-0.962690	-0.375253
Shane Watson	-0.952716	-3.997976	0.000000	-0.868611
Steve Smith	-0.833003	-6.062567	-4.729501	0.000000

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

$$\zeta_i = \frac{\sum_{j=1}^{m} \delta\left(A_i, A_j\right) - \delta_{\min}}{\delta_{\max} - \delta_{\min}}$$

where

$$\delta_{\min} = \min_i \sum_{j=1}^{m} \delta\left(A_i, A_j\right)$$

$$\delta_{\max} = \max_i \sum_{j=1}^{m} \delta\left(A_i, A_j\right)$$

and $i, j = 1, 2, \ldots, m$

[12]:
```
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'])
```

[12]:

	Sum
AB de Villiers	-27.726961
Andre Russel	-26.164804
Ben Stokes	-118.336927
Chris Gayle	-22.822275
Chris Lynn	-40.881867
David Warner	-5.721549
Faf Du Plessis	-43.175344
Jonny Bairstow	-30.788434
Jos Buttler	-70.209524
Kane Williamson	-103.617163
Kieron Pollard	-56.137062
Marcus Stoinis	-90.426875
Moeen Ali	-82.291213
Quinton de Kock	-25.554667
Shane Watson	-39.130688
Steve Smith	-61.068302

[13]:
```
delta_min = min(delta_sums)
delta_max = max(delta_sums)
pd.DataFrame(data=[delta_min, delta_max], columns=['Value'], index=['Minimum',␣
↪'Maximum'])
```

[13]:

	Value
Minimum	-118.336927
Maximum	-5.721549

[14]:
```
ratings = (delta_sums - delta_min) / (delta_max - delta_min)
pd.DataFrame(data=ratings, index=candidates, columns=['Rating'])
```

[14]:

	Rating
AB de Villiers	0.804597
Andre Russel	0.818468
Ben Stokes	0.000000
Chris Gayle	0.848149
Chris Lynn	0.687784
David Warner	1.000000

```
Faf Du Plessis   0.667418
Jonny Bairstow   0.777412
Jos Buttler      0.427361
Kane Williamson  0.130708
Kieron Pollard   0.552321
Marcus Stoinis   0.247835
Moeen Ali        0.320078
Quinton de Kock  0.823886
Shane Watson     0.703334
Steve Smith      0.508533
```

1.5 Step 4 - Create ranking based on the calculated ζ_i values

[15]:
```
def rank_according_to(data):
    ranks = (rankdata(data) - 1).astype(int)
    storage = np.zeros_like(candidates)
    storage[ranks] = candidates
    return storage[::-1]
```

[16]:
```
result = rank_according_to(ratings)
pd.DataFrame(data=result, index=range(1, m + 1), columns=['Name'])
```

[16]:

	Name
1	David Warner
2	Chris Gayle
3	Quinton de Kock
4	Andre Russel
5	AB de Villiers
6	Jonny Bairstow
7	Shane Watson
8	Chris Lynn
9	Faf Du Plessis
10	Kieron Pollard
11	Steve Smith
12	Jos Buttler
13	Moeen Ali
14	Marcus Stoinis
15	Kane Williamson
16	Ben Stokes

[17]:
```
print("The best candidate/alternative according to C* is " + result[0])
print("The preferences in descending order are " + ", ".join(result) + ".")
```

```
The best candidate/alternative according to C* is David Warner
The preferences in descending order are David Warner, Chris Gayle, Quinton de
Kock, Andre Russel, AB de Villiers, Jonny Bairstow, Shane Watson, Chris Lynn,
Faf Du Plessis, Kieron Pollard, Steve Smith, Jos Buttler, Moeen Ali, Marcus
```

Stoinis, Kane Williamson, Ben Stokes.