Analysis of IPL batting partnership networks

Jivjot Singh [*α]

[α] Department of Physical Sciences, Indian Institute for Science Education and Research Mohali- IISERM, Knowledge City, Manauli, 140306, Punjab, India

Abstract

This paper focuses on understanding the IPL-T20 cricket network using partnership statistics extracted by web-scraping. In the network, a partnership is shown by an edge between 2 players who are situated at the nodes. There is a sense of direction between the nodes, which confirms the better of the 2 batsmen during their partnership. This so called performance metric will give rise to the weight and direction to the edges. Finding out this metric and analysis of a directed network has been left for later. For now, I considered, if any two batsmen have batted together then they will share a edge and its weight would be 1; and so the network is **Undirected**. Confirming the fact that the resulting IPL batting network is a small-world network based on various centrality measures, I try to quantify a key/top-scoring batsman's ability to make partnerships.

Keywords: Web-scraping; Network science; Undirected network; Python; Beautifulsoup; Pandas; Networkx; Pyvis; Regex.

1 Introduction

The first step is the process of data collection. IPL data is widely available across major sports websites but ESPN has been used to fetch the statistics for the last 10 IPL seasons. To aggregate the partnership data, **Regular Expressions** are nested inside urls and class attributes in tandem with **BeautifulSoup**, a python package used to easily parse HTML content of webpages. After that **Dataframes** are constructed consisting of 3 columns - Names of both Batsmen and Weight of their Edge, making the finalized dataset ready for plotting network graphs and further analysis.

2 Python Script

The script.py includes everything from importing the necessary libraries to final datasets to plotting the network graphs and using inbuilt Networkx functions to analyse the structure. The code snippets shown below explain the process.

2.1 Web-Scraping using BeautifulSoup

```
url0='https://www.espncricinfo.com/ci/engine/series/index.html
        ?search=Indian+Premier+League;season=2022;view=
        season'
ipl0=rq.get(url0)
soup0=bs(ipl0.content,'html.parser')
yearlist=soup0.find_all('a',{'href':re.compile(r'.*season='+year+r'
        .*')})
yearsite=espn+ylist[0].get('href')
```

Here, I have used requests module to send HTTP requests and then used BeautifulSoup to fetch the HTML tags. Then to get the links for the desired IPL year, I have created regex patterns to find all the 'a' tags which have the specific 'href' attributes. This idea can be used to extend the same code to fetch all the match fixtures for each franchise team and then finally obtain the partnership statistics. (Note : For getting the desired data relevant to our objective it is imperative to study the HTML content of the website using 'Inspect' option.)

2.2 DataFrames using Pandas

```
team1_df=pd.DataFrame({'Batsman 1':b1list,'Batsman 2':b2list,'
        Weight':wtlist})
meandf=team1_df.groupby(['Batsman 1','Batsman 2'],as_index=
        False).agg (avg_wt= ('Weight', 'mean'))
```

After getting the data lists for both batsmen and their weights (1 or 0) throughout that ipl season, to plot the network graphs, I decide to convert these lists into 'DataFrames using 'Pandas'. To group the data for multiple entries of the same partnership, I used mean of the weights to get the final dataset.

```
result2018=pd.concat(frames)
result2019=pd.concat(frames)
result2020=pd.concat(frames)
result2021=pd.concat(frames)
result2022=pd.concat(frames)
colres=pd.concat([result2022,result2021,result2020,result2019,
        result2018])
```

This particular step could not be included in the loop for their have been different (state) teams throughout the last 10 seasons and also 1-2 teams had their franchise names changed.

2.3 Visualizing IPL Partnerships (2018-2022) using NetworkX

```
C=nx.from_pandas_edgelist(colres.groupby(['Batsman 1','
        Batsman 2'],as_index=False).agg (avg_wt= ('avg_wt', 'mean
        ')),source='Batsman 1',target='Batsman 2',edge_attr='
        avg_wt')
net=Network(notebook=True, bgcolor="#222222", font_color="
        white")
net.from_nx(C)
net.toggle_physics(True)
net.show('iplnetwork.html')
```

Getting the cumulative data of last 5 seasons enables for NetworkX to directly take the DataFrames as an input and

[*] ✉ ms18121@iisermohali.ac.in

give a network graph as output. This code creates a 'iplnetwork.html' file in the existing directory containing this script.

3 Analysis

3.1 Small-World Characteristics of IPL batting network

It is important for the cricket network to be a small world network, as our idea is to use the centrality measures and clustering coefficient to detect the key players in the network. However, most of the nodes are not directly connected in a small world, but the distance (i.e., number of hops) between two nodes is significantly less and the nodes connecting other nodes play significant role in the network.
In a network, small world coefficient σ can be defined as

$$\sigma = \frac{C_{Actual}/C_{Random}}{L_{Actual}/L_{Random}}$$

This value should be greater than 1 for being a small world network.
Here C_{Actual} and C_{Random} signify the clustering coefficients of players' network and random network respectively. Similarly, L_{Actual} and L_{Random} respectively signify the average path lengths of players network and random network.
My Results :
The number of nodes in my IPL network were 287; Also the $C_{Actual} = 0.421$ and $L_{Actual} = 1.467$. To quantify the Small-World Characteristics I have used the the quantity σ. One way to find it out is by generating a random network with the same number of nodes as the actual network (P. Erdos & A Renyi) and then finding out the clustering and average shortest path length coefficients for this random network. Luckily, NetworkX has inbuilt algorithm to find out this σ. For my analysis,

```
swcoeff = nx.sigma(C,niter=6,nrand=3)
```

This gives the small-world coefficient $3.07 > 1$. This result clearly depicts that IPL batting network exhibits small-world phenomenon.

3.2 Degree Correlation

There are 2 ways to analyse network's mixing pattern according to their degree correlation.
1. Correlation between a node's degree 'k' and its nearest neighbor's average degree 'k$_{nn}$'

```
knn=nx.k_nearest_neighbors(C)
plt.scatter(list(knn.keys()),list(knn.values()),s=5)
plt.xlabel(r'$k$',size=15)
plt.ylabel(r'$\langle k_{nn}(k) \rangle$',size=15)
z = np.polyfit(list(knn.keys()),list(knn.values()), 1)
line_function = np.poly1d(z)
plt.plot(list(knn.keys()), line_function(list(acoff.keys())), "y—",
        linewidth=1.5)
```

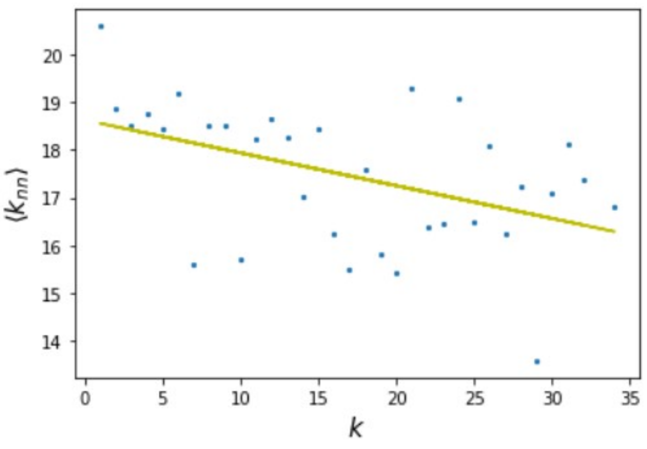

Figure 1: Average nearest-neighbor degree as a function of the node degree for the data of last 5 ipl seasons

The line fit depicts a slope of −0.06864
2. Assortativity Coefficient The assortative coefficient is also a good measure for a network's mixing pattern. Here, I have used the Pearson Correlation Coefficient to measure the degree assortativity of the graph.

```
acoff=nx.degree_pearson_correlation_coefficient(C)
```

This returns acoff = −0.05678. For $0 <$ acoff < 1, we have Assortative Mixing and $-1 <$ acoff < 0 implies Disassortative Mixing.
So the IPL batting network is **very weakly disassortative** in nature which is confirmed by both methods.

3.3 Centrality Measures

```
betcent=list(nx.betweenness_centrality(C).values())
clscent=list(nx.closeness_centrality(C).values())
eigcen=list(nx.eigenvector_centrality(C).values())
clus=list(nx.clustering(C).values())

betdf=pd.DataFrame({'Rank':degree,'betcent':betcent})
meanbetdf=betdf.groupby(['Rank'],as_index=False).agg (betcent=
        ('betcent', 'mean'))
clsdf=pd.DataFrame({'Rank':degree,'clscent':clscent})
meanclsdf=clsdf.groupby(['Rank'],as_index=False).agg (clscent=('
        clscent','mean'))
eigdf=pd.DataFrame({'Rank':degree,'eigcen':eigcen})
meaneigdf=eigdf.groupby(['Rank'],as_index=False).agg (eigcen=('
        eigcen','mean'))
cdf=pd.DataFrame({'Rank':degree,'clus':clus})
meancdf=cdf.groupby(['Rank'],as_index=False).agg (clus=('clus','
        mean'))
```

```
plt.scatter(meanbetdf.Rank,meanbetdf.betcent,s=70,marker='^',
        label='Betweenness Centrality')
plt.scatter(meanclsdf.Rank,meanclsdf.clscent,s=70,marker='x',
        label='Closeness Centrality')
plt.scatter(meaneigdf.Rank,meaneigdf.eigcen,s=70,marker='+',
        label='Eigenvector Centrality')
plt.scatter(meancdf.Rank,meancdf.clus,s=70,marker='d',label='
        Clustering Coefficient')

plt.grid()
plt.legend()
plt.xlabel('Degree')
plt.ylabel('Centrality(Normalized)')
plt.show()
```

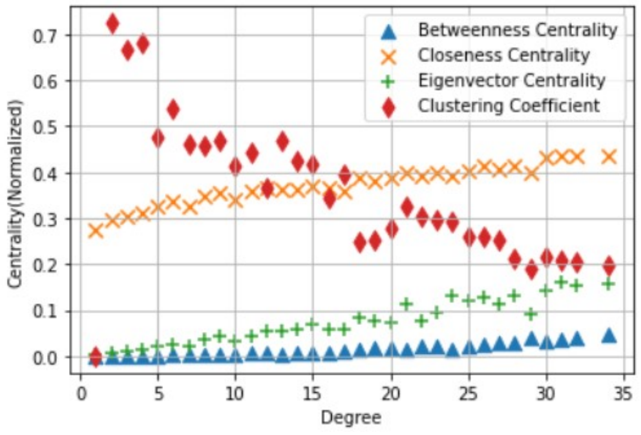

Figure 2: Different Normalized Centrality measures alongwith Clustering Coefficient vs Degree of node

1. Betweenness Centrality
Betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where $\mathbf{V}$ is the set of nodes, $\sigma(\mathbf{s},\mathbf{t})$ is the number of shortest (s,t)-paths, and $\sigma(\mathbf{s},\mathbf{t}|\mathbf{v})$ is the number of those paths passing through some node $\mathbf{v}$ other than $\mathbf{s},\mathbf{t}$. If $\mathbf{s}=\mathbf{t}$, $\sigma(\mathbf{s},\mathbf{t})=1$, and if $\mathbf{v} \in \mathbf{s},\mathbf{t}$, $\sigma(\mathbf{s},\mathbf{t}|\mathbf{v})=0$
2. Closeness Centrality
Closeness Centrality of a node **u** is the reciprocal of the average shortest path distance to **u** over all $\mathbf{n-1}$ reachable nodes

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$$

where $\mathbf{d}(\mathbf{v},\mathbf{u})$ is the shortest-path distance between **v** and **u**, and $\mathbf{n-1}$ is the number of nodes reachable from **u**. Notice that closeness distance function computes the incoming distance to **u** for directed graphs.
3. Eigenvector Centrality
Eigenvector centrality computes the centrality for a node based on the centrality of its neighbors. The eigenvector centrality for node **i** is the **i**-th element of the vector **x** defined by the equation

$$Ax = \lambda x$$

where **A** is the adjacency matrix of the graph **G** with eigenvalue λ. By virtue of the Perron–Frobenius theorem, there is a unique solution **x**, all of whose entries are positive, if λ is the largest eigenvalue of the adjacency matrix A
4. Clustering Coefficient
For unweighted graphs, the clustering of a node **u** is the fraction of possible triangles through that node that exist,

$$c_u = \frac{2T(u)}{deg(u)(deg(u)-1)}$$

where $\mathbf{T}(\mathbf{u})$ is the number of triangles through node **u** and $\mathbf{deg}(\mathbf{u})$ is the degree of **u**.
For weighted graphs, there are several ways to define clustering. The one used here is defined as the geometric average of the subgraph edge weights.

$$c_u = \frac{1}{deg(u)(deg(u)-1)} \sum_{vw} (\hat{w}_{uv}\hat{w}_{uw}\hat{w}_{vw})^{1/3}$$

The edge weights $\hat{\mathbf{w}}_{\mathbf{uv}}$ are normalized by the maximum weight in the network $\hat{\mathbf{w}}_{\mathbf{uv}} = \mathbf{w}_{\mathbf{uv}}/\mathbf{max}(\mathbf{w})$. The value of $\mathbf{c}_{\mathbf{u}}$ is assigned to 0 if $\mathbf{deg}(\mathbf{u}) < 2$.

3.4 Best Batsmen

This subsection presents a very rudimentary idea of how to go about deciding the skim of batsmen based on the most degree. This idea can be extended to the previously mentioned idea of a complex performance metric and a concept of direction in a partnership which depends on a few different parameters.

```
bbp=sorted(dict(C.degree).items(), key=lambda item: item[1],
     reverse=True)
bb=[]
for i in range(len(bbp)):
    if bbp[i][1]>=np.percentile(bblinks,99):
        bb.append(bbp[i][0])
```

This returns the top 5 batsmen : ['GJ Maxwell', 'DJ Hooda', 'SS Iyer', 'V Kohli', 'R Tewatia']
I have decided to consider the top 1% of the degree distribution as threshold to conclude the best batsmen out of the 2018-2022 IPL seasons; considering only the ability to form partnerships. Now, the point to note is that this idea does not take into account whether these batsmen scored comparatively higher than the rest. This list just shows these batsmen have been more reliable.

B	EV	C
GJ Maxwell	MP Stoinis	R Tewatia
KD Karthik	GJ Maxwell	GJ Maxwell
R Tewatia	RR Pant	MP Stoinis
DJ Hooda	R Tewatia	DA Miller
MP Stoinis	SS Iyer	SS Iyer
SS Iyer	DJ Hooda	KL Rahul

Table 1: Best 6 Batsmen Line-up w.r.t different centrality measures: B-Betweenness Centrality, EV-Eigenvector Centrality, C-Closeness Centrality

4 Conclusion

The focus of this paper is to aggregate the partnership statistics using Web-Scraping and then plotting Network Graphs using NetworkX. Based on small-world analysis, I have tried to come up with top order batsmen who have been essentially crucial in forming partnerships based on the last 5 IPL seasons. These key players with high centrality values have an impact on their respective franchise teams' performance. I have also found out that the IPL Network is very weak Disassortative type using the ANND (Average Nearest Neighbor Degree) analysis.

References

Dey, P., M. Ganguly, and S. Roy (2017). "Network centrality based team formation: A case study on T-20 cricket". *Applied computing and informatics* 13(2), pages 161–168.

Erdos, P., A. Rényi, et al. (1960). "On the evolution of random graphs". *Publ. Math. Inst. Hung. Acad. Sci* 5(1), pages 17–60.

Hao, D. and C. Li (2011). "The dichotomy in degree correlation of biological networks". *PloS one* 6(12), e28322.

Hardiman, S. J., P. Richmond, and S. Hutzler (2009). "Calculating statistics of complex networks through random walks with an application to the on-line social network Bebo". *The European Physical Journal B* 71(4), pages 611–622.

Vázquez, A. (2003). "Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations". *Physical Review E* 67(5), page 056104.

Zhou, S. and G.-Q. Zhang (2007). "Chinese Internet AS-level topology". *IET communications* 1(2), pages 209–214.