The PageRank Algorithm and its application to IPL cricket player's rankings

Kushagra Agrawal, Simran Barnwal
Department of Electrical and Computer Engineering
University of California, San Diego
k4agrawal@ucsd.edu, sbarnwal@ucsd.edu

Abstract

The aim of the project is to study the PageRank algorithm from a Linear Algebra perspective. This algorithm is used by Google to rank webpages given a search term. Broadly, it's a rating algorithm that ranks nodes in a directed graph given links (with or without weights) between nodes. Over the course of the project, we shall examine the applications of linear algebra in the algorithm and some of its major variations, and study related topics like Markov Chains. Finally, we will implement the algorithm in Python to rank cricket players based on their performance.

1 Importance of the topic

The PageRank algorithm, is perhaps, one of the most influential algorithms that has shaped the current landscape of information retrieval. First proposed by Lawrence Page and Sergey Brin [3], the algorithm became the foundation of Google search, the unquestionable leader in the search-engine space. Discussions about the applications of PageRank are often limited to web-based querying and search results. However, at its heart, PageRank is simply a ranking algorithm that tries to determine the most important nodes in a graph. It does so using key Linear algebra concepts of Markov chains, transition matrix and Eigen values. We believe that studying the PageRank algorithm from a mathematical perspective will help us derive deeper insights into the algorithm and gain a better understanding of the mathematics of rankings. To further facilitate our learning, we plan on using the PageRank algorithm to determine the best cricket players in the IPL based on their history in the tournament. We plan on creating a player graph with links related to factors such as runs scored, wickets taken, number of dismissals etc. and running PageRank on the graphs to determine the best players in the IPL.

2 Implementation

IPL, or the Indian Premier League, is an annual cricket tournament where players from all over the world compete as part of ten cricket teams. Similar to soccer leagues in Europe, the players in IPL also switch teams between seasons through "player auctions". The best players are always in high demand and are the most sought after by the team owners. To judge rankings or capabilities of players, we can use their previous performance to create a weighted graph over which we can use the PageRank algorithm to rank the players. This can help the team owners decide the value a player might bring to the team, and what might be a good bid for a player.

We will refer the IPL Complete dataset from 2008 to 2020 [1] on Kaggle.

2.1 Brief overview of proposed graph

A popular team sport, cricket broadly consists of two kinds of players, batsmen and bowlers. A game typically consists of two innings with each team alternating between batting and bowling. The team that scores the most runs at the end of the match wins. The role of a batsman is to score runs while a bowler's job is to prevent that and dismiss the batsman. The Kaggle dataset [1] captures the information of each match through columns related to ball number, bowler who bowled it, batsman facing, venues of the matches, toss results etc. The dataset can be effectively utilised to create weighted graphs.

We plan on using PageRank algorithm on two weighted graphs, one of the bowlers and one of the batsmen. Applying a similar method to the PageRank algorithm, Govan et al [2] developed a method called the GEM method, using the margin of victory v1-v2 to weight the edge between two football teams, where v1 and v2 are the teams' scores against each other in the NFL. We shall implement a similar algorithm with a different metric for the weights of the edge.

Specifically, we shall determine the ranking among batsmen by creating a graph, with an edge between two batsmen signifying the difference between the runs scored against common bowlers. There will be a similar graph among bowlers, where the edge between two bowlers can be defined as the inverse of the difference in runs each has conceded to common batsmen. The exact definition of the link weight will be defined later during the course of the project.

3 Timeline

Table 1: Timeline.

31st January, 2022	Study Markov Chains and its use in PageRank
7th February, 2022	Theoretical analysis of PageRank
14th February, 2022	Define the formulation of the algorithm's implementation on the dataset
16th Februray, 2022	Midterm Report Submission
21st february, 2022	Exploratory Data Analysis on the dataset
28th February, 2022	Preprocessing the dataset, creating the graph
7th March, 2022	Implementing the algorithm on the graph created
17th March, 2022	Final Results and comparisons
18th March, 2022	Final Report submission

4 Markov Chain Theory

4.1 Problem Formulation and Definitions

Markov processes are stochastic processes describing a sequence of possible events where the probability of each event depends solely on the state achieved in the previous event. In other words, conditional on the present state of the system, the past and future states are independent. Classic examples of markov chains include movement of stock prices and dynamics of animal populations.

A markov process with either discrete state space or discrete indices are often called markov chains. In this project, we deal with markov chains that contain a finite number of states. The transitions in a markov chain are defined as the changes in the state of the system, with the probabilities associated with each change stored in a transition matrix. Therefore, for a system $\mathbf{x_k} \in \mathbb{R}^n$,, the core principle of Markov chain can be stated as follows:

$$\mathbf{x_{k+1}} = T(\mathbf{x_k}), \quad \text{for time } k = 0, 1, 2... \tag{1}$$

where $T : \mathbb{R}^n \to \mathbb{R}^n$. is the Transition matrix we described earlier.

Markov chains are near ubiquitous, featuring prominently in forecasting, information theory and economics. The next section illustrates the applications of markov chains by predicting population distribution.

4.2 Example of Markov Process

Suppose that in 2000, the population of a city is 600,000 and the population of its suburbs is 400,000. We are also given that in any year, 1) 5% of people move to the suburbs, and 2) 3% of the people move to the city. If we want to find the population distribution in 2060 assuming the change of population is restricted solely with the movements of peoples, we can treat the problem as a markov process.

Using the above information, the transition matrix can be formulated as follows:

$$\begin{array}{rcc} & \text{From City} & \text{From Suburbs} \\ \text{To City} & .95 & .03 \\ \text{To Suburbs} & .05 & .97 \end{array} \tag{2}$$

$$A = \begin{bmatrix} .95 & .03 \\ .05 & .97 \end{bmatrix}. \tag{3}$$

We can convert the population to a probability vector by simply normalizing the population distribution as follows:

$$\mathbf{x_0} = \frac{1}{1,000,000} \begin{bmatrix} 600,000 \\ 400,000 \end{bmatrix} = \begin{bmatrix} 0.60 \\ 0.40 \end{bmatrix}. \tag{4}$$

Using the Markov property, The population distribution in 2001 will simply be the product of transition matrix A and $\mathbf{x_0}$ as follows:

$$\mathbf{x_1} = A\mathbf{x_0} = \begin{bmatrix} .95 & .03 \\ .05 & .97 \end{bmatrix} \begin{bmatrix} 0.60 \\ 0.40 \end{bmatrix} = \begin{bmatrix} 0.582 \\ 0.418 \end{bmatrix}. \tag{5}$$

Similarly, for 2002, we can write it as a product of A and $\mathbf{x_1}$, where $\mathbf{x_1}$ itself is $A\mathbf{x_0}$. Using this, we can estimate the population distribution for 2060, i.e., $k = 60$ as:

$$\mathbf{x_{60}} = \overbrace{A \cdots A}^{60} \mathbf{x_0} = A^{60}\mathbf{x_0}. = \begin{bmatrix} 0.3766 \\ 0.6233 \end{bmatrix} \tag{6}$$

That is, only 37% of the population remains in the city.

4.3 Steady State of Markov processes

We can further extend our discussion to show how the population distribution doesn't change a lot after a certain point, i.e. it reaches a "steady state". Mathematically, a steady vector $\mathbf{q}$ is defined for a stochastic matrix P such that $P\mathbf{q} = \mathbf{q}$. This, in fact, is a unique property of markov chains that can be encapsulated by the theorem below:

We shall make and prove the following claims in the theorems below: If P is an $n \times n$ regular stochastic matrix, then P has a unique steady-state vector $\mathbf{q}$. Further, if $\mathbf{x_0}$ is any initial state and $\mathbf{x_{k+1}} = P\mathbf{x_k}$ for $k = 0, 1, 2, \ldots$, then the Markov Chain $\{\mathbf{x_k}\}$ converges to $\mathbf{q}$ as $k \to \infty$.

Here, a stochastic matrix P is regular if some matrix power P^k contains only strictly positive values. The above property highlights a remarkable property of Markov Chains: it converges to a steady-state vector irrespective of the state the chain starts in. It also reinforces the core property of a Markov Chain: the long-term behavior of the chain has no memory of the starting state.

We can visualise the steady-state of our example below (for code, see Appendix):

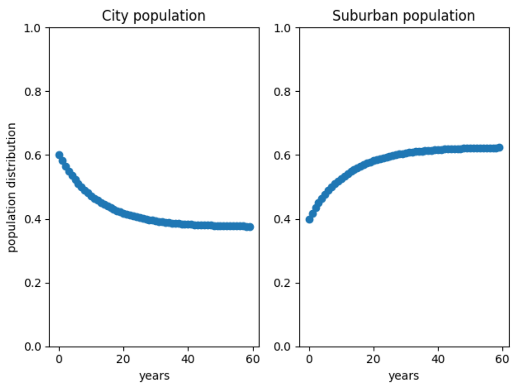

Figure 1: Population distribution over the years

The above graph illustrates that the population distribution doesn't seem to change much after $k = 40$, i.e. it has reached its steady state. It also satisfies the property $P\mathbf{q} = \mathbf{q}$. as highlighted below

$$\begin{bmatrix} .95 & .03 \\ .05 & .97 \end{bmatrix} \begin{bmatrix} .38 \\ .61 \end{bmatrix} = \begin{bmatrix} .38 \\ .61 \end{bmatrix}. \tag{7}$$

The following theorems and proofs will help us understand the above properties better.

Theorem 1. *(Perron-Frobenius eigenvalue) For any strictly positive matrix $A > 0$ there exist $\lambda_0 > 0$ and $x_0 > 0$ such that*

1. $A * x_0 = \lambda_0 * x_0$
2. *if $\lambda \neq \lambda_0$ is any other eigenvalue of A, then $|lambda| < \lambda_0$*
3. *λ_0 has geometric and algebraic multiplicity 1.*

Proof.

1. Let λ_0 be as defined above, and let $x_0 \geq 0$ be a vector such that $A * x_0 \geq \lambda_0 * x_0$. We shall prove this by contradiction. Let's assume that $A * x_0 \neq \lambda_0 * x_0$, i.e. that $A * x_0 \gtrsim \lambda_0 * x_0$, but not $A * x_0 = \lambda_0 * x_0$. Consider the vector $y_0 = A * x_0$. Since $A > 0$, $A * x > 0$ for any $x \gtrsim 0$. Specifically,
$$A(y_0 - \lambda_0 * x_0) = A * y_0 - \lambda_0 * y_0 > 0$$
i.e. $A * y_0 > \lambda_0 * y_0$; but this contradicts our definition of λ_0. Consequently,$A * x_0 = \lambda * x_0$, and furthermore $x_0 = \frac{1}{\lambda} A * x_0 > 0$.

2. Let $\lambda \neq \lambda_0$ be an eigenvalue of A and $y \neq 0$ the corresponding eigenvector, $A * y = \lambda * y$. Denote $|y| = (|y_1|, ..., |y_n|)$. Since $A > 0$, it is the case that
$$A * |y| \geq |A * y| = |\lambda * y| = |\lambda| * |y|$$
. By the definition of λ_0, it follows that $|\lambda| \leq \lambda_0$. To prove strict inequality, let $\delta > 0$ be so small that the matrix $A * \delta = A - \delta * I$ is still strictly positive. Then $A * \delta$ has eigenvalues $\lambda_0 - \delta$ and $\lambda - \delta$, since $(\lambda - \delta) * I - A * \delta = \lambda * I - A$. Furthermore, since $A * \delta > 0$, its largest eigenvalue is $\lambda_0 - \delta$, so $|\lambda - \delta| \leq \lambda_0 - \delta$. This implies that A can have no eigenvalues $\lambda \neq \lambda_0$ on the circle $|\lambda| = \lambda_0$, because that would have $|\lambda - \delta| > |\lambda_0 - \delta|$.

3. We shall consider only the geometric multiplicity. Let there be another (real) eigenvector $y > 0$, linearly independent of x_0, associated to λ_0. Then one could form a linear combination $w = x_0 + \alpha * y$ such that $w \gtrsim 0$, but not $w > 0$. However, since $A > 0$, it must be the case that also $w = \frac{1}{\lambda} A * w > 0$

□

Theorem 2. *Let $A > 0$ be a strictly positive $n \times n$ matrix with row and column sums $r_i = \sum_i a_{ij}$, $c_j = \sum_j a_{ij}$, where $i, j \in [1,n]$ Then for the "Perron-Frobenius eigenvalue" λ_0 of Theorem 1 the following bounds hold:* $\min_i r_i \leq \lambda_0 \leq \max_i r_i$, $\min_j c_j \leq \lambda_0 \leq \max_j c_j$

Proof. Let $x_0 = (x_1, x_2, ..., x_n)$ be an eigenvector corresponding to λ_0, normalised so that $\sum_i x_i = 1$. Summing up the equations for $A * x_0 = \lambda_0 * x_0$ yields:

$$a_{11} * x_1 + a_{12} * x_2 + ... + a_{1n} * x_n = \lambda_0 * x_1$$

$$a_{21} * x_1 + a_{22} * x_2 + ... + a_{2n} * x_n = \lambda_0 * x_2$$

and so on till

$$a_{n1} * x_1 + a_{n2} * x_2 + ... + a_{nn} * x_n = \lambda_0 * x_n$$

Adding the LHS and RHS together we get:

$$c_1 * x_1 + c_2 * x_2 + ... + c_n * x_n = \lambda_0 * (x_1 + x_2 + ... + x_n)$$

Thus, λ_0 is a weighted average of the column sums, hence, $\min_j c_j \leq \lambda_0 \leq \max_j c_j$. We can make a similar argument for A^T which will have the same eigenvalue λ_0 and say, $\min_i r_i \leq \lambda_0 \leq \max_i r_i$.

□

Theorem 3. *Let $P \leq 0$ be the transition matrix of a regular Markov chain. Then there exists a unique distribution vector π such that $\pi * P = \pi$.*

Proof. By Theorem 1, P has a unique largest eigenvalue $\lambda_0 \in \mathbf{R}$. By Theorem 2, $\lambda_0 = 1$, because as a stochastic matrix all col sums of P are 1. Since the geometric multiplicity of λ_0 is 1, there is a unique stochastic vector π (i.e. satisfying $\sum_i \pi_i = 1$) such that $\pi * P = \pi$.

□

5 PageRank Algorithm

5.1 Problem Formulation

Let W := $\{wp_1, wp_2, ..., wp_n\}$ be a set of webpages that are retrieved for a particular query. We represent the results as a directed graph $G = (V, E)$ where $V = W$ and E = set of directed edges where an edge (wp_i, wp_j) symbolizes that wp_i has a link to wp_j. Given graph $G = (V, E)$, find a ranking/importance of the webpages W, such that the more important webpages have a higher probability of the user visiting it amongst all webpages in W. The more important webpages should be displayed first to reflect relevant answers to the user's query.

5.2 Algorithm Description

5.2.1 Defining importance of a webpage

A webpage is important if many other important webpages have a link to it. This can be represented as the following equation:

$$Importance(wp_i) = \sum_{j=1}^{n} Importance(wp_j) * probability((wp_j, wp_i))$$

where $probability((wp_j, wp_i))$ refers to the probability of going from webpage wp_j to webpage wp_i.

An important thing to note here is that this equation is self-referential, i.e. the importance of webpages appear on both sides of the equation. To solve this equation we need to assign a fixed importance to each webpage. However, the above equation can be represented as the following $x = Px$ where x can be thought of a vector representing the importance of each of the webpages and P can be thought of as the transition matrix. Here is where the concept of random walk can be applied to PageRank. As we have studied in the convergence proofs of Markov Chains, we know that there exists a unique distribution of x called π such that $\pi * P = \pi$.

5.2.2 Brief Introduction to the Algorithm

Now that we understand how the concept of Markov Chains and random walks is introduced in the PageRank algorithm, let's explore it in a little more detail. As mentioned by Larry Page and Sergey Brin in their paper [3], PageRank models user behaviour. Let's assume we have a "random surfer". The random surfer may start on any webpage at random and keeps clicking the links randomly without clicking the back button. The probability that the random surfer lands on a webpage wp_i can be defined as the PageRank or importance of the webpage wp_i.

The above formulation helps us break down the algorithm into the following specific steps:

1. Create a graph G from the set of webpages W retrieved for a particular query
2. Create a Markov chain that formulates a random walk on the graph G (find the transition matrix P)
3. Find the steady state of the given Markov Chain, order the webpages according to their probabilities (importances) in the found steady state.

As defined in the section on Markov chains, only a regular transition matrix has a steady state vector. An important task is to convert the initial transition matrix P to a regular transition matrix P'. This can be done in the following ways:

1. **Handling dangling nodes:** We call a node in graph G as a dangling node if it has no outgoing edges. Thus, the column corresponding to this webpage in P will be a zero column. We know that such a matrix cannot be regular, since for any P^k the k^{th} column will always be zero. To handle this, we look at the problem formulation of PageRank: "the random surfer may start on any webpage at random". It is reasonable to assume that if the random surfer arrives at a webpage with no outgoing links, it chooses among all webpages in W at random. Thus, we can replace the zero column by a column with all entries as $1/n$ where n is $|W|$. We do this operation for all dangling nodes in graph G.
2. **Making certain that the matrix P is regular:** Even after this change we cannot be certain that for an arbitrary set of webpages the matrix P will still be regular. We go back to the problem formulation for a solution: the random surfer might get bored and start on another random webpage. This implies that there is a small probability α that the surfer will jump from a page to any other page at random. We can't just add α to all entries in the matrix P because all columns must sum to 1. Instead, we will multiply all entries by (1-α) and add a factor α/n to it. Thus, the columns still sum to 1. To formalize:

$$P = (1-\alpha)P + \alpha/n$$

   where n is $|W|$.

The above changes make the matrix P regular because all of its entries are positive, with the minimum entries being α/n where n is $|W|$. Now, any positive integer power of P, P^k will be such that all entries in P^k will be positive. Thus, according to Theorem 3, we are certain now that given this transition matrix, we have a steady state for the Markov Chain and a unique vector π as described in Theorem 3.

To find this vector π, a simplistic method is to find the eigenvalues and eigenvectors of the regular transition matrix. However, this method has a time complexity of $O(n^3)$ where n is the number of webpages. To make this faster we use the power method, which relies on the convergence of the steady state vector. Since we have a transition matrix which is regular, convergence is guaranteed (i.e. the Markov Chain has a steady state vector). The power method is used to find the dominant eigenvalue and the corresponding eigenvector of a matrix (given it exists, which we made sure of). It can be summarized as below:

1. To apply the power method to a square matrix P, we begin with an initial guess for the eigenvector with the dominant eigenvalue ($\lambda = 1$)

2. We multiply the most recent value of the eigenvector on the left by P, and normalize the result to obtain a new eigenvector.
3. We keep doing it until the eigenvector converges. The eigenvector thus obtained is the π we are looking for called the steady state eigenvector corresponding to $\lambda = 1$

5.2.3 Pseudocode

The algorithm described above can be more formally stated by the following pseudocode:

Algorithm 1 Function to create transition matrix given webpages and their outlinks

```
function CREATETRANSITIONMATRIX(W)
    P := []                                  ▷ Matrix: P will be the initial transition matrix for graph G
    for $wp_i \in W$ do
        outlinks_i := $|links((wp_i, wp_j))|$          ▷ Number of links originating from $wp_i$
        for $wp_j \in W$ do
            if $link(wp_i, wp_j)$ then
                P[j,i] := 1/outlinks_i
            end if
        end for
    end for
    return P
end function
```

Algorithm 2 Function to make a transition matrix regular by assuming random surfer model

```
function MAKETRANSITIONMATRIXREGULAR(W, P, alpha)
    for $wp_i \in W$ do
        if $|links((wp_i, wp_j))|$==0 then
            P[:,i]=1/$|W|$                                  ▷ Handling dangling nodes
        end if
    end for
    $P = (1 - alpha) * P + alpha/|W|$
    return P
end function
```

Algorithm 3 The main PageRank algorithm

```
function PAGERANK(W, alpha)
    P := CREATETRANSITIONMATRIX(W)
    P := MAKETRANSITIONMATRIXREGULAR(W, P, alpha)        ▷ Convert P to a regular matrix
    v = 1/n * ones((n,1))
    while v has not converged do                        ▷ Power Method
        v = v*P
        v_norm = norm(v)
        v= v/v_norm
    end while
    return SORT(W, key = v)
end function
```

As mentioned before, we shall apply the concepts covered in the previous sections to rank players in the Indian Premier League, based on their performances from 2008 to 2020. We will use a modified version of PageRank where the edges of the graphs are assigned certain weights. This "weighted" PageRank, we believe, will serve as a better representation of player's capabilities and help make a more informed decision while ranking players.

We plan on creating two graphs, one each for bowlers and batsmen, in order to rank them. Let B_i and B_j denote two players in our batsmen graph $G^1 = (B, E^1, w^1)$ where B is the set of all batsmen as vertices. We will use the following metrics to quantify a batsmen's calibre:

- Average runs scored by the batsmen
- Runs scored by B_i and B_j against a common bowler Bo_k
- Count of dismissals by common bowlers against B_i and B_j
- Count of catches taken by the batsmen B_i and B_j as fielders.

The edge weight w^1_{ij} of edge E^1_{ij} between B_i and B_j will simply be the difference of the weighted sum of these metrics. Thus, our graph G^1 would have edges between each of the vertices in B. On similar lines, let Bo_i and Bo_j denote two players in our bowler graph $G^2 = (Bo, E^2, w^2)$ where Bo is the set of bowlers. We will use the following metrics to quantify a bowler's calibre:

- Average runs conceded by the bowlers
- Count of dismissals by two bowlers Bo_i and Bo_j against a common batsman B_k
- Count of invalid balls bowled by bowlers (no-balls, wides etc.)
- Count of catches taken by bowlers Bo_i and Bo_j as fielders.

The edge weight w^2_{ij} of edge E^2_{ij} between Bo_i and Bo_j will simply be the difference of the weighted sum of these metrics. Running the weighted PageRank algorithm on both the graphs separately should give us the desired rankings of batsmen and bowlers.

6 Experiments

As mentioned in the previous section, we applied weighted PageRank to the IPL dataset to determine the top 25 batsmen and bowlers using the criterion given above.

6.1 Rankings of batsmen and bowlers

To assign the weights of edges of the graph for batsmen, we used the following metrics to quantify a batsmen's calibre:

- Average runs scored by the batsmen (weight 0.3)
- Runs scored by B_i and B_j against a common bowler Bo_k (weight 0.3)
- Count of dismissals by common bowlers against B_i and B_j (weight 0.3)
- Count of catches taken by the batsmen B_i and B_j as fielders.(weight 0.1)

Similarly, we used the following metrics to quantify a bowler's calibre:

- Average runs conceded by the bowlers (weight 0.2, reverse normalized as inversely proportional to it)
- Count of average wickets taken by bowler (weight 0.3)
- Count of dismissals by two bowlers Bo_i and Bo_j against common batsmen B_k (weight 0.2, reverse normalized as inversely proportional to it)
- Count of runs conceded by two bowlers Bo_i and Bo_j against common batsmen B_k (weight 0.2, reverse normalized as inversely proportional to it)
- Count of catches taken by bowlers as fielders. (weight 0.1)

Upon creating the P matrix and applying weighted PageRank to the above graphs, we obtain the rankings below. Upon researching further, we observed that some of the top players predicted by the page rank algorithm were some who haven't played in recent years. For example, Sohail Tanvir was the best bowler predicted by the page rank algorithm. However, the last he had played was in 2008 where he was the leading wicket-taker which explains his high rank. Since the algorithm takes the average metrics for players over matches, some players who haven't played in the recent years and played only for a few seasons really well were favoured over the others. In order to get more updated rankings we consider only the matches held after 2015 from here on. The top 25 batsmen and bowlers calculated by the PageRank algorithm over the period 2015-2020 are as follows:

Ranks	Batsmen	Bowlers
1	DA Warner	L Ngidi
2	KL Rahul	BE Hendricks
3	AB de Villiers	MA Starc
4	V Kohli	K Rabada
5	JM Bairstow	AS Joseph
6	Q de Kock	KW Richardson
7	HM Amla	KK Ahmed
8	CH Gayle	Sachin Baby
9	RR Pant	AJ Tye
10	F du Plessis	Rashid Khan
11	JC Buttler	A Zampa
12	LMP Simmons	JO Holder
13	S Dhawan	JC Archer
14	TM Head	A Nehra
15	KS Williamson	NM Coulter-Nile
16	D Padikkal	YS Chahal
17	CA Lynn	GB Hogg
18	RD Gaikwad	JJ Bumrah
19	SPD Smith	CH Morris
20	MK Pandey	S Lamichhane
21	AM Rahane	TM Head
22	MS Dhoni	KMA Paul
23	JP Duminy	Imran Tahir
24	G Gambhir	A Nortje
25	SV Samson	O Thomas

Table 2: Players rankings

6.1.1 Qualitative evaluation of PageRank results

To judge whether our rankings our accurate, we looked at the average IPL statistics of the highest paid cricketers in IPL.

Among the top 10 batsmen predicted by our algorithm, most of the batsmen like V Kohli, DA Warner, and RR Pant are among the highest paid cricketers in IPL for the year 2021 showing that the page rank's predictions are valid. Similarly, for the bowlers, cricketers like KW Richardson were some of the highest paid bowlers in IPL 2021.[5]

6.2 How does the eigenvector v change over iterations

Upon analyzing the value of the transition matrices over iterations, we notice that the values of the v eigenvector for the top 10 players starts converging after approximately 8 iterations, as illustrated below:

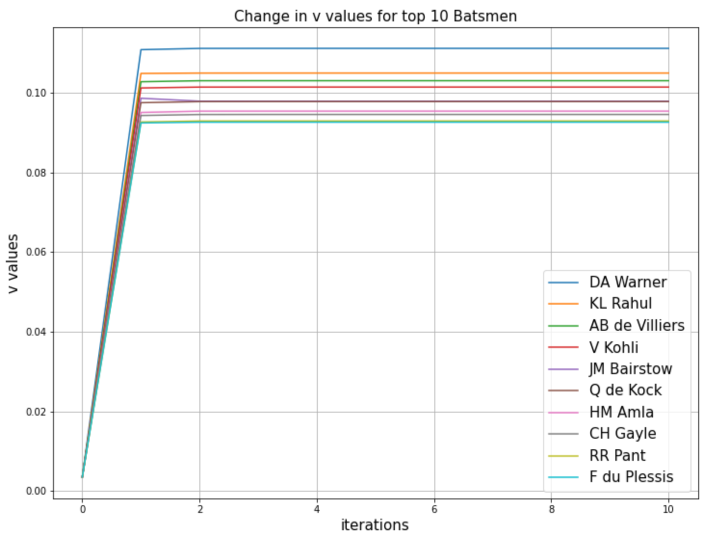

Figure 2: The change in v values of the top 10 batsmen over the number of iterations

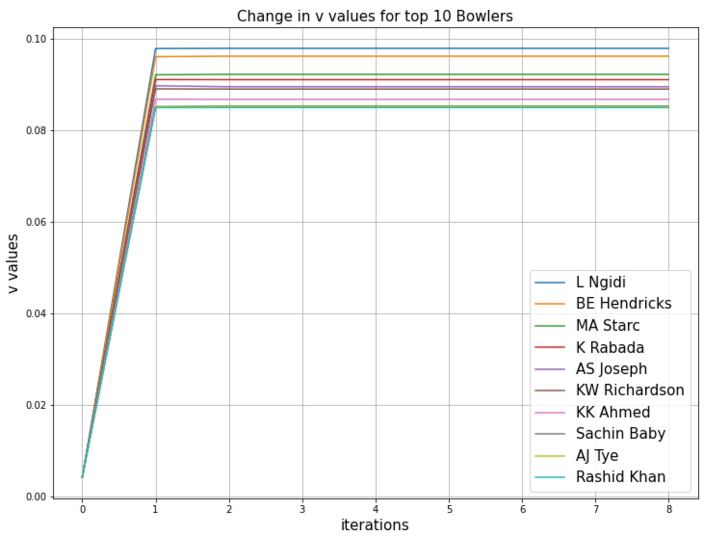

Figure 3: The change in v values of the top 10 bowlers over the number of iterations

We see that initially the v values (the probability of landing at a webpage in page rank algorithm by a random surfer) for these players is negligible (1/287 for batsmen and 1/231 for bowlers). However, as the number of iterations increases, the v values for these players increases to substantial amounts of 0.1 i.e. the probability of landing at the player increases significantly.

6.3 Convergence of eigenvector v

Utilising the power method to compute the eigenvector v, our PageRank algorithm iteratively computes the ranking eigenvector. As highlighted by the example above, the eigenvector converges to a steady state after a few iterations. We plot the norm of the difference of the intermediate eigenvector v to its steady value over the number of iterations. We observe that the difference in norm decreases over the number of iterations and v converges to a steady state, proving the validity of the PageRank algorithm for our use case.

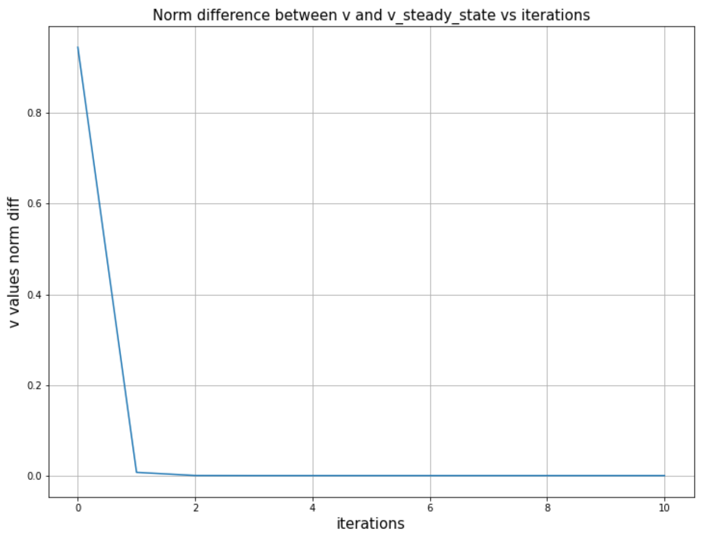

Figure 4: The change in norm of v w.r.t. it's steady state over the number of iterations for the batsmen P matrix eigenvector

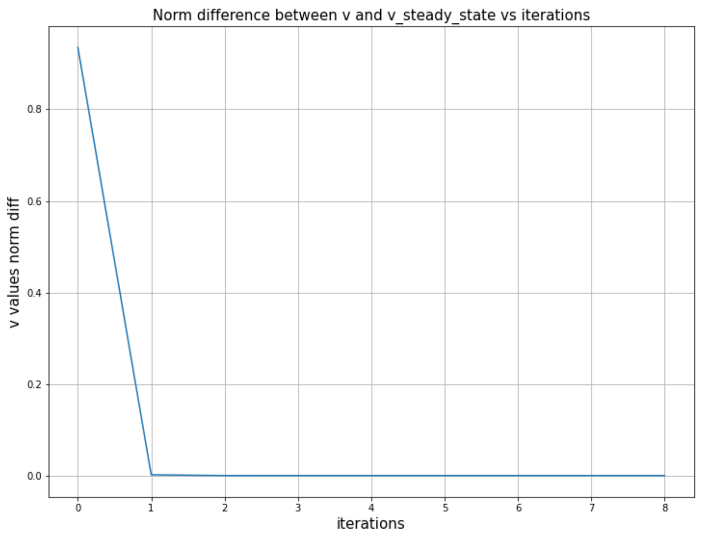

Figure 5: The change in norm of v w.r.t. it's steady state over the number of iterations for the bowlers P matrix eigenvector

7 References

[1] IPL Complete dataset from 2008 to 2020: https://www.kaggle.com/patrickb1912/ipl-complete-dataset-20082020

[2] A. Y. Govan, C. D. Meyer, and R. Albright, “Generalizing Google’s PageRank to rank National Football League teams”, in Proceedings of the SAS Global Forum, SAS Global Users Group/SAS Institute, Cary, NC, 2008.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web”, tech report, Stanford InfoLab, 1999, http://ilpubs.stanford.edu:8090/422/.

[4] Proofs regarding steady state of Markov Chain: http://www.tcs.hut.fi/Studies/T-79.250/tekstit/lecnotes_01.pdf

[5] Players salaries in IPL 2021: https://www.business-standard.com/article/sports/ipl-auction-2021-live-updates-full-players-list-team-available-purse-121021800479_1.html

8 Appendix

8.1 Python script to simulate Markov process for Population Distribution

```
import numpy as np
import matplotlib.pyplot as plt

# initial transition matrix A
A = np.array(
    [[0.95,0.03],
     [0.05,0.97]])

# initial state vector x
x = np.array([0.60,0.40])

xs = np.zeros((60,2))

# compute future state vectors
for i in range(60):
    xs[i] = x
    print(f'x({i}) = {x}')
    x = A @ x

P*x = plt.subplot(121)
plt.plot(range(60),xs.T[0],'o-')
P*x.set_ylim([0,1])
plt.title('City population',size=12)
plt.xlabel('years')
plt.ylabel('population distribution')
P*x = plt.subplot(122)
plt.plot(range(60),xs.T[1],'o-')
P*x.set_ylim([0,1])
plt.title('Suburban population',size=12)
plt.xlabel('years')
plt.tight_layout()
plt.savefig('graphs.png')
```