Erick Silvestre Lima de Brito;
Germano Andrade Brandão;

Algoritmo do Pagerank

Rio de Janeiro
2020

FGV EMAp
ESCOLA DE MATEMÁTICA APLICADA

Sumário

Introdução

Este trabalho visa a mostrar o funcionamento do algoritmo do Pagerank do ponto de vista da Álgebra Linear. Pretende-se introduzir a metodologia por trás do algoritmo e também aplicações em base de dados reais.

1 Surgimento do Pagerank

1.1 Motivação

No início dos anos 90, quando os primeiros motores de busca usavam *sistemas de classificação baseados em texto* para decidir quais páginas eram mais relevantes para uma determinada consulta, o que o mecanismo de pesquisa fazia era manter um índice de todas as páginas da web, e quando um usuário digitava sua consulta, o mecanismo navegava por seu índice e contava as ocorrências das palavras-chave em cada arquivo da web. As vencedoras eram as páginas com maior número de ocorrências das palavras-chave. Elas seriam exibidas de volta para o usuário.

No entanto, houve uma série de problemas com esta abordagem. Uma pesquisa sobre um termo comum como "Internet" era problemática. Suponha que decidimos escrever um site que contenha a palavra "FGV" um bilhão de vezes e nada mais. Caso um usuário digitasse essa palavra em um mecanismo de pesquisa, como tudo o que o mecanismo faz é contar as ocorrências das palavras fornecidas na consulta, muito provavelmente o nosso site seria o primeiro a ser exibido, embora não fosse relevante para este indivíduo.

A utilidade de um mecanismo de pesquisa depende da relevância do conjunto de resultados que ele fornece. É claro que pode haver milhões de páginas da web que incluem uma palavra ou frase específica; no entanto, alguns deles serão mais relevantes, populares ou confiáveis do que outros. O usuário não tem capacidade ou paciência para percorrer todas as páginas que contêm as palavras de consulta fornecidas. Espera-se que as páginas relevantes sejam exibidas entre as 20-30 páginas principais retornadas pelo mecanismo de pesquisa.

1.2 Criação

Os mecanismos de pesquisa modernos empregam métodos de classificação dos resultados para fornecer os "melhore" resultados primeiro, que são mais elaborados do que apenas classificação em texto simples.

O algoritmo Page Rank usado pelo mecanismo de busca Google é um dos algoritmos mais conhecidos e influentes para calcular a relevância das páginas da web.

Foi inventado por **Larry Page** e **Sergey Brin** quando eles eram estudantes de graduação em Stanford, e se tornou uma marca registrada do Google em 1998.

1.3 Propósito

A ideia que o Page Rank trouxe era que, a importância de qualquer página da web pode ser avaliada olhando para as páginas que apontam (*linkam*) para ela. Se criarmos uma página da web i e incluirmos um hiperlink para a página da web j , isso significa que consideramos j importante e relevante para o nosso tópico. Se houver muitas páginas vinculadas a j , isso significa que a crença comum é que a página j é importante.

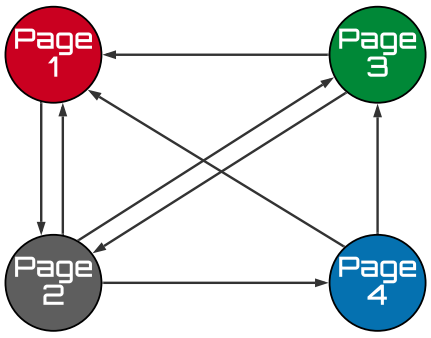
Se, por outro lado, j tem apenas um backlink, mas vem de um site autorizado k , (como www.google.com, www.bbc.com, emap.fgv.br) dizemos que k transfere sua autoridade para j ; em outras palavras, k afirma que j é importante. Uma vez que estamos falando sobre popularidade ou hierarquia, podemos atribuir iterativamente uma classificação a cada página da web, com base nas classificações das páginas que apontam para ela.
E *voilà*! Temos nosso **Page Rank**.

2 Demonstração do Funcionamento

Para fins de demonstração, começamos retratando a rede da web como um grafo direcionado, com nós representados por páginas da web e arestas representadas pelos links entre elas.

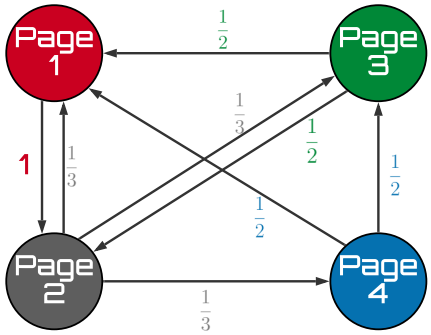
Suponha, por exemplo, que temos uma pequena Internet composta de apenas 4 sites www.page1.com, www.page2.com, www.page3.com, www.page4.com, referenciando uns aos outros por meio de hiperlinks.
Vamos "traduzir" essa Internet em um gráfico direcionado com 4 nós, um para cada página.

Quando o site i faz referência a j , adicionamos uma aresta direcionada entre o nó i e o nó j, no grfo. Para calcular a classificação da página, ignoramos todos os links de navegação, como os botões voltar e avançar, pois nos preocupamos apenas com as conexões entre diferentes sites. Por exemplo, a **Page 2** se vincula a todas as outras páginas, de modo que, no grafo, o nó que a representa terá arestas de saída para todos os outros nós. A **Page 1** tem apenas um link, para a **Page 2**, portanto, o nó que a representa terá uma aresta de saída para o nó da **Page 2**. Depois de analisar cada página da web, obtemos o seguinte grafo:



Em nosso modelo, cada página deve transferir uniformemente sua importância para as páginas às quais se vincula. O nó 2 tem 3 arestas de saída, então ele passará sua importância para cada um dos outros 3 nós. O nó 1 possui apenas uma aresta de saída, então ele passará toda sua importância para o nó 2.

Em geral, se um nó tiver k arestas de saída, ele passará $\frac{1}{k}$ de sua importância para cada um dos nós aos quais se vincula. Vamos visualizar melhor o processo atribuindo pesos a cada aresta.



Denotemos por A a matriz de transição do grafo, $A = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & 0 & 0 \end{bmatrix}$

2.1 Onde entra a Álgebra Linear

O primeiro ponto a se destacar é que sempre teremos A como uma **Matriz de Markov** (ou estocástica), que é definida como uma matriz não negativa e todas as colunas somam 1. Ou seja,

$$a_{ij} \geq 0 \ , \forall i, j$$

e também

$$\sum_{i=1}^{n} a_{ij} = 1 \ , \forall j \in \{1, \ldots, n\}$$

Calculando a importância da matriz A, pelo método da potência, com um vetor inicial $v = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$, ao atualizarmos a classificação de cada página adicionando ao valor atual a importância dos links de entrada, isso é equivalente a multiplicar a

matriz A por pelo vetor v. Fazendo isso até termos uma convergência para o vetor estacionário, ficamos com:

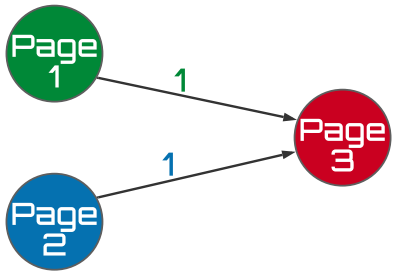
```
A^1v = [0.33 0.38 0.21 0.08]
A^2v = [0.27 0.44 0.17 0.12]
A^3v = [0.29 0.35 0.21 0.15]
A^4v = [0.3  0.4  0.19 0.12]
A^5v = [0.29 0.39 0.19 0.13]
A^6v = [0.29 0.38 0.2  0.13]
A^7v = [0.29 0.39 0.19 0.13]
A^8v = [0.29 0.39 0.19 0.13]
```

Ou seja, nosso vetor de Pagerank é $v^* = \begin{bmatrix} 0.29 \\ 0.39 \\ 0.19 \\ 0.13 \end{bmatrix}$.

O vetor Page Rank v^* que calculamos por métodos diferentes indica que a página 2 é a página mais relevante. Isso pode parecer surpreendente, pois a página 2 tem 2 backlinks, enquanto a página 1 tem 3 backlinks. Se dermos uma olhada no gráfico, vemos que o nó 1 tem apenas uma aresta de saída para o nó 2, então ele transfere toda a sua importância para o nó 2. Equivalentemente, uma vez que um internauta que segue apenas os hiperlinks visita a página 1, ele só poderá ir para a página 2.

3 Problemas

3.1 Nós sem arestas de saída



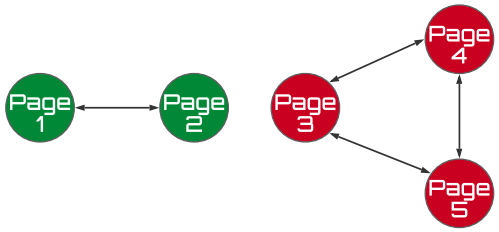
Calculando iterativamente a classificação das 3 páginas:

$$v_0 = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}, v_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2/3 \end{bmatrix}, v_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 2/3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Portanto, neste caso, a classificação de cada página é 0. Isso é contra-intuitivo, já que a página 3 tem 2 links de entrada, então ela deve ter alguma importância!

Uma solução fácil para esse problema seria substituir a coluna correspondente ao nó pendente 3 por um vetor de coluna com todas as entradas $\frac{1}{3}$. Desta forma, a importância do nó 3 seria igualmente redistribuída entre os outros nós do grafo, ao invés de ser perdida.

3.2 Componentes desconectados



Um surfista aleatório que começa no primeiro componente conectado não tem como chegar à página 5 da web, pois os nós 1 e 2 não têm links para o nó 5 que ele possa seguir. A álgebra linear também não ajuda. A matriz de transição para este gráfico é $A = \left[\begin{array}{cc|ccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 1/2 & 0 \end{array}\right]$.

Observe que $v = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ e $u = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ ambos são autovetores correspondentes ao autovalor 1 e não são apenas trivialmente um ou um múltiplo escalar um do outro. Portanto, tanto na teoria quanto na prática, a notação das páginas de classificação do primeiro componente conectado em relação às do segundo componente conectado é ambígua.

4 A solução de Page e Brin

A web é muito heterogênea por natureza, e certamente enorme, então não esperamos que seu gráfico esteja conectado. Da mesma forma, haverá páginas totalmente descritivas e sem links de saída. O que deve ser feito neste caso? Precisamos de um significado não ambíguo para a classificação de uma página, para qualquer gráfico da Web direcionado com n nós.

Para superar esses problemas, fixe uma constante positiva p entre 0 e 1, que chamamos de fator de amortecimento (um valor típico para p é 0,15). Defina a matriz de **Page Rank** (também conhecida como matriz Google) do gráfico por Matriz PageRank onde $M = (1-p)\cdot A + p\cdot B$, onde $B = \frac{1}{n}\cdot \begin{bmatrix} 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$.

Vamos provar que a matriz M é de Markov, ou seja, que

$$\sum_{i=1}^{n} m_{ij} = 1 \ , \ \forall j \in \{1, \dots, n\}$$

Sabemos que

$$\sum_{i=1}^{n} m_{ij} = \sum_{i=1}^{n} ((1-p)\cdot a_{ij} + p\cdot b_{ij}) = \sum_{i=1}^{n}((1-p)\cdot a_{ij}) + \sum_{i=1}^{n} p\cdot b_{ij}$$

$$= (1-p)\sum_{i=1}^{n} a_{ij} + p\sum_{i=1}^{n} b_{ij} = (1-p)\cdot 1 + p\cdot 1 = 1 - \cancel{p} + \cancel{p} = 1$$

Intuitivamente, a matriz M "conecta" o gráfico e elimina os nós pendentes. Um nó sem arestas de saída agora tem probabilidade de $\frac{p}{n}$ se mover para qualquer outro nó. Rigorosamente, para a matriz M , os seguintes teoremas se aplicam:

Teorema de Perron-Frobenius: Se M é uma matriz estocástica de coluna positiva, então:

1. 1 é um autovalor de multiplicidade um.
2. 1 é o maior autovalor: todos os outros autovalores possuem valor absoluto menor que 1.
3. Os autovetores correspondentes ao autovalor 1 têm apenas entradas positivas ou apenas entradas negativas. Em particular, para o autovalor 1 existe um único autovetor com a soma de suas entradas igual a 1.

Teorema de Convergência do Método de Potência: Seja M uma matriz $n \times n$ estocástica de coluna positiva . Denote por v^* seu autovetor probabilístico correspondente ao autovalor 1. Seja z o vetor coluna com todas as entradas iguais a $\frac{1}{n}$. Então a sequência $z, Mz, \ldots, M^k z$ converge para o vetor v^*.

Fato: O vetor PageRank para um gráfico da web com matriz de transição A e fator de amortecimento p é o único autovetor probabilístico da matriz M , correspondendo ao autovalor 1.

5 Aplicação em uma base de dados

A base escolhida trata sobre o Campeonato Brasileiro de Futebol das temporadas de 2013 a 2020. Ela contém todas as partidas realizadas e, dentre outros dados, os nomes dos times, data da partida, ano, total de gols de cada time, vencedor da partida, etc.

[1]:
```
#Importando pacotes necessários
import pandas as pd
import numpy as np
```

[2]:
```
#Importando base de dados
br = pd.read_csv("../Databases/BRA.csv")
```

[3]:
```
#Criando uma função para filtrar a base por ano (temporada) e␣
↪turno.
#Também escolhemos algumas colunas mais interessantes para a␣
↪gente.

def filtrar_base(ano, turno):
    filtro = br["Season"] == ano
    if turno == 1:
        return br[filtro][["Home", "Away", "HG", "AG", "Res"]].
↪head(190)
    elif turno == 2:
        return br[filtro][["Home", "Away", "HG", "AG", "Res"]].
↪tail(190)
```

Agora, vamos filtrar a base, por exemplo, apenas pelo primeiro turno do ano de 2017

[4]:
```
br = filtrar_base(2017, 1)
```

e ficamos com

[5]:
```
br.head()
```

[5]:

	Home	Away	HG	AG	Res
1900	Flamengo RJ	Atletico-MG	1.0	1.0	D
1901	Corinthians	Chapecoense-SC	1.0	1.0	D
1902	Fluminense	Santos	3.0	2.0	H
1903	Avai	Vitoria	0.0	0.0	D
1904	Bahia	Atletico-PR	6.0	2.0	H

Nesse momento, estamos rumo à criação da matriz.
Primeiramente, vamos criar um *DataFrame* apenas com as colunas de ***Winner*** e ***Loser***.

[6]:
```
winner = []
loser = []

for index, row in br.iterrows():
        if row['Res'] == "H":
                winner.append(br.loc[index, 'Home'])
                loser.append(br.loc[index, 'Away'])
        elif row['Res'] == "A":
                winner.append(br.loc[index, 'Away'])
                loser.append(br.loc[index, 'Home'])

        elif row["Res"] == "D":
                winner.append(br.loc[index, 'Home'])
                loser.append(br.loc[index, 'Away'])

                winner.append(br.loc[index, 'Away'])
                loser.append(br.loc[index, 'Home'])
```

Feito isso, temos as colunas da nossa tabela apenas com os vencedores e perdedores em cada coluna.

[7]:
```
#Criando DataFrame
w_l = pd.DataFrame({"Winner": winner, "Loser": loser})

w_l.head()
```

[7]:

	Winner	Loser
0	Flamengo RJ	Atletico-MG
1	Atletico-MG	Flamengo RJ
2	Corinthians	Chapecoense-SC
3	Chapecoense-SC	Corinthians
4	Fluminense	Santos

Nesse ponto, estamos começando a desenhar a nossa matriz de transição

[8]:
```
# 'index' se refere aos times distintos que participaram dessa␣
 ↪edição do brasileirão.
# ela já está em ordem alfabética
index = sorted(w_l["Winner"].unique())

#Criação do DataFrame que servirá como matriz de transição
matriz = pd.DataFrame(index=index, columns=index)

#Mostrando uma submatriz 5x5 com apenas as primeiras 5 linhas e␣
 ↪5 colunas da matriz.
matriz.head()[matriz.columns[:5]]
```

[8]:

	Atletico GO	Atletico-MG	Atletico-PR	Avai	Bahia
Atletico GO	NaN	NaN	NaN	NaN	NaN
Atletico-MG	NaN	NaN	NaN	NaN	NaN
Atletico-PR	NaN	NaN	NaN	NaN	NaN
Avai	NaN	NaN	NaN	NaN	NaN
Bahia	NaN	NaN	NaN	NaN	NaN

Criando uma função que verifica se o time da coluna perdeu para o time que está a linha.
Caso isso ocorra, o **NaN** é substituido pelo número 1.

[9]:
```
def verifica_time(valor, coluna):

        for cada_indice in w_l.index:
                if w_l.loc[cada_indice, "Loser"] == coluna and␣
 ↪w_l.loc[
                    cada_indice, "Winner"] == valor.name:
                        return 1

#Criando matriz de transição apenas com 1 e 0
```

```
transicao = pd.DataFrame()
for i in matriz:
        transicao[i] = matriz.apply(verifica_time, args=(i, )).
 ↪fillna(0)

transicao.head()[transicao.columns[:5]]
```

[9]:

	Atletico GO	Atletico-MG	Atletico-PR	Avai	Bahia
Atletico GO	0.0	0.0	0.0	1.0	0.0
Atletico-MG	1.0	0.0	0.0	1.0	0.0
Atletico-PR	1.0	1.0	0.0	1.0	0.0
Avai	0.0	0.0	0.0	0.0	1.0
Bahia	1.0	1.0	1.0	1.0	0.0

Nesse ponto, podemos finalizar nossa matriz **A**, dividindo cada coluna pela sua soma

[10]:
```
finalizada = pd.DataFrame()
for i in transicao:
        finalizada[i] = transicao[i] / sum(transicao[i])

finalizada.head()[finalizada.columns[:5]].round(3)
```

[10]:

	Atletico GO	Atletico-MG	Atletico-PR	Avai	Bahia
Atletico GO	0.000	0.000	0.000	0.067	0.000
Atletico-MG	0.062	0.000	0.000	0.067	0.000
Atletico-PR	0.062	0.077	0.000	0.067	0.000
Avai	0.000	0.000	0.000	0.000	0.077
Bahia	0.062	0.077	0.083	0.067	0.000

Finalmente, podemos transformar o *DataFrame* para um *array* **numpy**, ou seja, transformá-la em um formato de matriz como estamos habituados.

[11]:
```
A = finalizada.to_numpy()
```

Como visto, para melhores resultados, podemos calcular a matriz $M = (1-p) \cdot A + p \cdot B$, onde $B = \frac{1}{n} \cdot \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$ e p é escolhido igual a 0.15.

[12]:
```
p = 0.15
n = A.shape[0]

B = (1/n)*np.ones((n, n))
M = (1-p)*A + p*B
```

[13]:
```
#Essa função retorna uma tupla contendo dois `np.arrays`, onde o
#primeiro se refere aos autovalores e o segundo, aos autovetores␣
 ↪correspondentes.
w, v = np.linalg.eig(M)

#Autovetor correspondente ao autovalor 1
vetor_pagerank = v[:, 0].real

#Tabela contendo o nome dos times e o pagerank de cada um.
pagerank = pd.DataFrame({
    "Times": index,
    "Pagerank": np.absolute(vetor_pagerank)
})

#Ordenando a tabela de acordo com o pagerank
pagerank.sort_values(by="Pagerank", ascending=False, inplace=True)

pagerank
```

[13]:

	Times	Pagerank
7	Corinthians	0.329069
10	Flamengo RJ	0.283475
15	Santos	0.254503
12	Gremio	0.247502
3	Avai	0.246925
9	Cruzeiro	0.245660
2	Atletico-PR	0.238486
11	Fluminense	0.237158
17	Sport Recife	0.228049
8	Coritiba	0.218695
6	Chapecoense-SC	0.216743
5	Botafogo RJ	0.216452
13	Palmeiras	0.210694
14	Ponte Preta	0.203577

1	Atletico-MG	0.196003
4	Bahia	0.179209
18	Vasco	0.173636
16	Sao Paulo	0.172670
19	Vitoria	0.160491
0	Atletico GO	0.123012

Agora, filtrando pelo ano de 2018:

[14]:
```
br = filtrar_base(2018, 1)
```

[14]:

	Times	Pagerank
11	Gremio	0.289166
13	Palmeiras	0.287783
16	Sao Paulo	0.285372
12	Internacional	0.274504
9	Flamengo RJ	0.256528
6	Chapecoense-SC	0.245159
10	Fluminense	0.218257
1	Atletico-MG	0.212538
3	Bahia	0.211568
17	Sport Recife	0.209950
4	Botafogo RJ	0.207756
8	Cruzeiro	0.206139
2	Atletico-PR	0.200230
5	Ceara	0.200108
18	Vasco	0.196409
7	Corinthians	0.192012
0	America MG	0.184698
14	Parana	0.184136
15	Santos	0.182909
19	Vitoria	0.162806

E então, pelo ano de 2019:

[15]:
```
br = filtrar_base(2019, 1)
```

[15]:

	Times	Pagerank
8	Corinthians	0.296163
16	Palmeiras	0.294798
18	Sao Paulo	0.293707

3	Bahia	0.277608
10	Flamengo RJ	0.277288
15	Internacional	0.263889
14	Gremio	0.249776
17	Santos	0.247148
19	Vasco	0.206318
9	Cruzeiro	0.193982
5	CSA	0.193178
1	Atletico-MG	0.190953
6	Ceara	0.188077
0	Athletico-PR	0.186296
11	Fluminense	0.180202
12	Fortaleza	0.171392
4	Botafogo RJ	0.169045
2	Avai	0.166602
13	Goias	0.161079
7	Chapecoense-SC	0.154503