Tarea 12

Reyes García Andrea,
Rivera Hernández León Diego.

Jueves, 21 de Enero 2021

1. Realizar el ejercicio 5 de la práctica 12, con los ejemplos de cadenas mostrados en la práctica. Además, encontrar las distribuciones invariantes, si es posible, utilizando un programa para calcular eigenvalores y eigenvectores (usando la paquetería de Python). Comparar las respuestas.

   Solución:

```
import numpy as np

A1 = np.array([1/2, 1/2, 0, 1/2, 0, 1/2, 0, 1/2, 1/2]).reshape(3,3);
A2 = np.array([1/2, 1/2, 0, 1/2, 99/200, 1/200, 0, 0, 1]).reshape(3,3);
A3 = np.array([0,1,0,0,0,0,
                1/5,0,4/5,0,0,0,
                0,2/5,0,3/5,0,0,
                0,0,3/5,0,2/5,0,
                0,0,0,4/5,0,1/5,
                0,0,0,0,1,0]).reshape(6,6);
A4 = np.array([1,0,0,0,0,0,
                0.5,0,0.5,0,0,0,
                0,0.5,0,0.5,0,0,
                0,0,0.5,0,0.5,0,
                0,0,0,0.5,0,0.5,
                0,0,0,0,0,1]).reshape(6,6);

# Para la primera matriz:
e1, P1 = np.linalg.eig(A1);
P1inv = np.linalg.inv(P1);
D1 = np.diag(e1);
A1n = np.matmul(P1,np.matmul(np.linalg.matrix_power(D1,100000),P1inv)); # (no se
    tarda nada en hacer 100,000 multiplicaciones)
Pi0 = np.array([0.4,0.25,0.35]);
PI1 = np.matmul(Pi0, A1n); # Distribución invariante


# Para la segunda matriz:
e2, P2 = np.linalg.eig(A2);
P2inv = np.linalg.inv(P2);
D2 = np.diag(e2);
A2n = np.matmul(P2,np.matmul(np.linalg.matrix_power(D2,100000),P2inv));
Pi0 = np.array([0.24,0.36,0.4]);
PI2 = np.matmul(Pi0, A2n);


# Para la tercera matriz:
e3, P3 = np.linalg.eig(A3);
P3inv = np.linalg.inv(P3);
D3 = np.diag(e3);
A3n = np.matmul(P3,np.matmul(np.linalg.matrix_power(D3,100000),P3inv));
Pi0 = np.array([0.14,0.25,0.15,0.12,0.14,0.2]);
PI3 = np.matmul(Pi0, A3n);

# Para la cuarta matriz:
e4, P4 = np.linalg.eig(A4);
P4inv = np.linalg.inv(P4);
```

```
D3 = np.diag(e4);
A4n = np.matmul(P4,np.matmul(np.linalg.matrix_power(D4,100000),P4inv));
Pi0 = np.array([0.14,0.25,0.15,0.12,0.14,0.2]);
PI4 = np.matmul(Pi0, A4n);
```

Conclusiones: Se pudo observar que solo la primera matriz tiene distribuciones invariantes.

Esto se debe a que cumple la propiedad:

v es invariante si para $\{x_n\}$ si $vP = v$, es decir, si

$$v(i) - \sum_{i \in S} v(i)p(i,j)$$

para todo $j \in S$

En el primer caso, la distribución invariante fue: $\pi = (1/3, 1/3, 1/3)$

2. En este ejercicio vamos a hacer modelación con una versión muy simplificada del algoritmo original del Page Rank de Google. Tomemos una base de datos modesta proveniente de Twitter.

   Los usuarios (o páginas) representan los vértices y el "seguir" (o hipervínculos dirigidos) es una arista dirigida (la punta es la persona seguida y la cola el seguidor).

   (a) Construya la gráfica dirigida asociada, y hada que Python dibuje la gráfica (visualice vértices y aristas). Haga eso también con una subgráfica (subconjunto de vértices y aristas de la gráfica original) para visualizarlo mejor.

   (b) Considere un proceso estocástico X_n dado por una caminata aleatoria simple en la gráfica:

   - El caminante inicia en un vértice inicial i_0 seleccionado de forma uniforme sobre todos los vértices de la gráfica.
   - Dado que al tiempo n el caminante se encuentra en el vértice i (es decir $X_n = i$), el caminante elige una de las aristas **salientes** del vértice de forma equiprobable y llega a un nuevo vértice j, entonces ocurre $X_{n+1} = j$.

   Note que el proceso anterior podría tener el problema de que en un momento podría quedarse atrapado: si, no hay aristas salientes, no se puede continuar (recuerde que Taylor Swift no sigue a nadie en Instagram). Entonces es necesario añadir la regla siguiente:

   - Si en un momento n el vértice no tiene aristas salientes, el caminante es teletransportado aleatoriamente: se elige un vértice j al azar entre todos los vértices de la gráfica y al siguiente tiempo $n + 1$ el caminante sigue desde tal vértice j.

   (c) Simule el proceso anterior en Python. Ahora, para cada vértice i y tiempo n defian el número:

$$N(i,n) = \frac{1}{n} \sum_{k=0}^{n} \mathbb{1}_i(X_k),$$

   es decir, la proporción de veces que el vértices i fue visitado por el caminante en el intervalo de tiempo $\{0, 1, ..., n\}$. Haga que su programa compute ese número para cada vértice, dado que el caminante hizo n pasos totales.

   (d) La lista PageRank simplemente ordena a los vértices de acuerdo a su valor $N(i,n)$, de mayor a menor. Entre más grande sea el número, más importante es el sitio. Imprima tal lista, habiendo elegido n como el número más grande que la computadora lo permita.

   (e) Grafique de nuevo la gráfica asociada a la base de Twitter, pero haga que cada nodo tenga tamaño proporcional a $N(i,n)$. Entre más importante sea el usuario o sitio se debe ver más gordo el vértice.

   (f) Explique claramente:

   i. ¿Cuáles son las fortalezas y cuáles las debilidades del algoritmo? (matemáticas, computacionales y de aplicación al mundo real).

ii. ¿Cómo podría manipularse el algortimo, como sitio (o sitios) interesado(s) en tener un alto PageRank?

iii. Como diseñador del algoritmo interesado en la justicia, ¿cómo podría combatirse la manipulación anterior?

iv. Mencione al menos dos variaciones factibles del algoritmo y sus mejoras o pérdidas respecto a este algoritmo.

Solución:

A continuación, presentamos el programa con el que simulamos y graficamos todo lo solicitado:

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

# Creamos la gráfica a partir de los datos:
G = nx.read_edgelist('Twitter_network_R.csv', delimiter = ',', create_using = nx
    .DiGraph(), nodetype = str);

usuarios = np.array(G.nodes());
adyacencia = nx.to_numpy_matrix(G);

# Vamos a contar seguidores y seguidos de cada usuario:
seguidores = [sum([adyacencia[i,j] for i in range(34)]) for j in range(34)];
seguidos = [sum([adyacencia[i,j] for j in range(34)]) for i in range(34)];

# Creamos la matriz de transición a partir de la de adyacencia y las
    indicaciones de la tarea:
transicion = np.zeros((34,34));
for i in range(34):
  sumafila = sum([adyacencia[i,k] for k in range(34)]);
  print(i);
  print(sumafila);
  for j in range(34):
    if sumafila == 0:
      transicion[i,j] = 1 / 34; # Si el vértice no tiene aristas salientes, el
    caminante es teletransportado a cualquier nodo de la gráfica.
    if adyacencia[i,j] == 1 and sumafila != 0:
      transicion[i,j] = 1 / sumafila;

# Simulamos la cadena de Markov:
Xn = [];
ultimo = list(np.random.choice(a = range(34),size = 1))[0]; # Asumimos
    distribución inicial uniforme.
for i in range(100000):
  Xn.append(usuarios[ultimo]);
  ultimo = list(np.random.choice(a = range(34),size =  1, p = list(transicion[
    ultimo,:])))[0];

# Contamos:
conteo = [Xn.count(usuarios[i]) for i in range(34)];

################################### Gráficos ###################################

# Gráfica:
nx.draw(G, with_labels = 1, node_color = 'steelblue', edge_color = '
    mediumturquoise');
plt.show();

# Subgráfica:
subG = G.subgraph(usuarios[5:25]);
nx.draw(subG, with_labels = 1, node_color = 'limegreen', edge_color = '
    springgreen');
plt.show();

```

```
# Histograma:
plt.hist(Xn, bins = 34, color = 'gold', ec = 'yellow');
plt.title('X_n');
plt.ylabel('Frecuencia');
plt.xticks(rotation = 'vertical');
plt.show();

# Gráfica con tamaño de nodos por importancia:
nx.draw(G, with_labels = 1, node_size = conteo, node_color = 'hotpink',
    edge_color = 'pink');
plt.show();
```

Después de realizar la simulación, presentamos los resultados obtenidos de ella en la siguiente tabla:

Usuario	Importancia	$N(i,n)$	Seguidores	Seguidos
@isdotr	1°	12,799	33	6
@hadleywickham	2°	12,726	29	8
@revodavid	3°	9,478	28	16
@amandacox	4°	5,591	6	2
@eddelbuettel	5°	5,500	19	7
@robjhyndman	6°	5,108	18	14
@adamlaiacano	7°	4,545	6	5
@brendan642	8°	4,509	11	3
@bshor	9°	3,338	5	4
@berndweiss	10°	3,055	8	15
@freakonometrics	11°	2,981	17	23
@genetics_blog	12°	2,811	8	4
@statsinthewild	13°	2,549	17	22
@piboonrungroj	14°	2,360	17	15
@stat110	15°	2,086	17	18
@rickwicklin	16°	2,014	14	17
@cjbayesian	17°	1,650	17	24
@kobriendublin	18°	1,638	11	17
@gswithr	19°	1,529	10	10
@gdequeiroz	20°	1,462	11	18
@_inundata	21°	1,384	10	8
@statsepi	22°	1,193	10	10
@drago_carlo	23°	1,075	9	16
@kristin_linn	24°	1,019	6	5
@statbandit	25°	993	10	26
@econo_stats	26°	946	8	4
@prisonrodeo	27°	880	2	4
@abmathewks	28°	864	8	14
@jjgibaja	29°	846	6	14
@emilopezcano	30°	763	8	14
@biff_bruise	31°	755	8	13
@treycausey	32°	584	5	7
@cboettig	33°	527	4	8
@tomtensfeldt	34°	442	4	9

Cuadro 1: Tabla con los resultados de la simulación. Se tomó $n = 100,000$.

Finalmente, mostramos recursos gráficos que serán de ayuda (figuras 1 a 4):

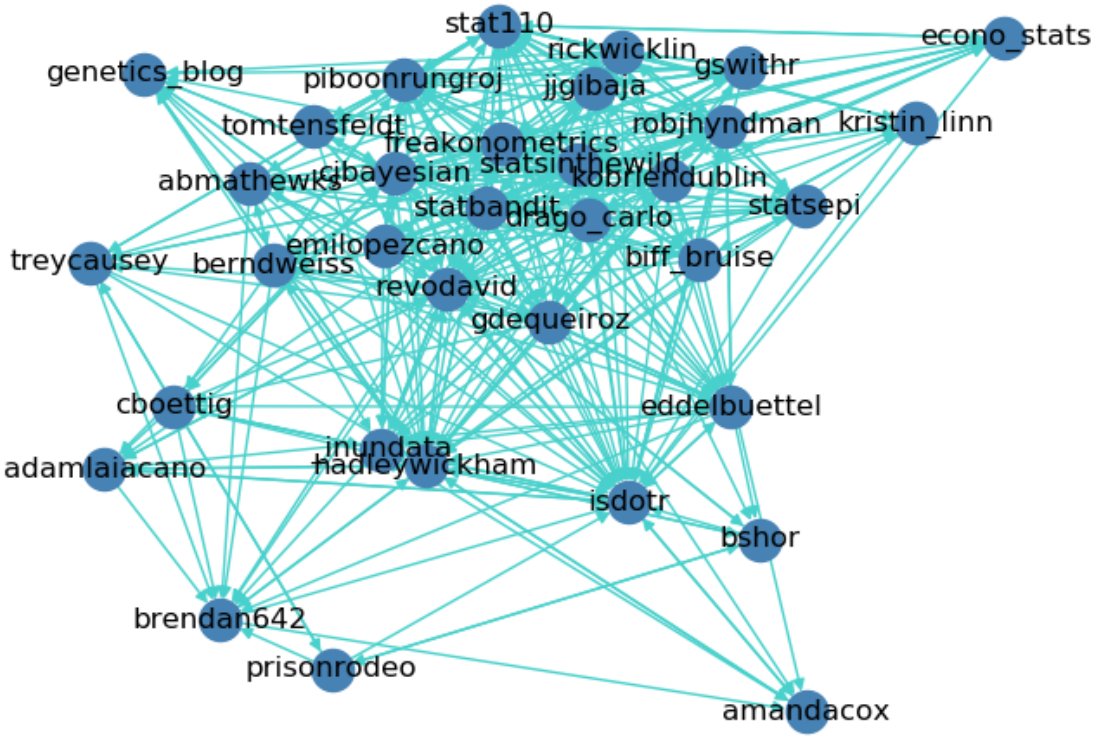

Figura 1: Gráfica dirigida con los usuarios de Twitter considerados. La relación $A \to B$ indica: A sigue a B.

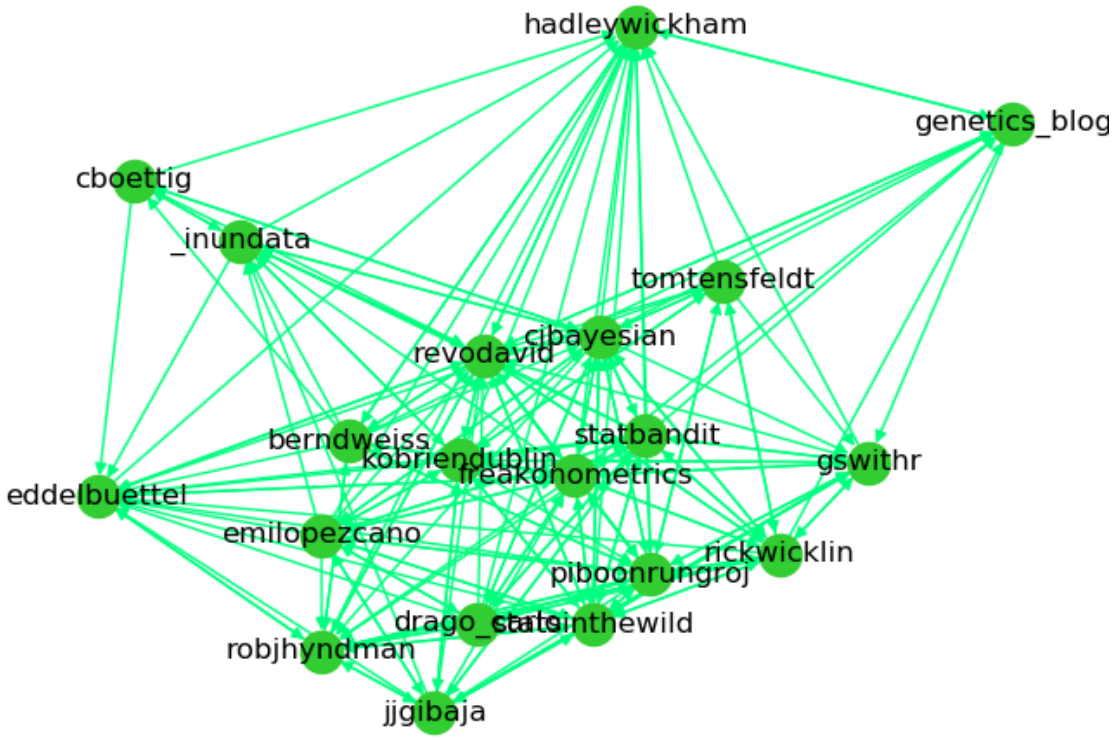

Figura 2: Subgráfica dirigida de la original.

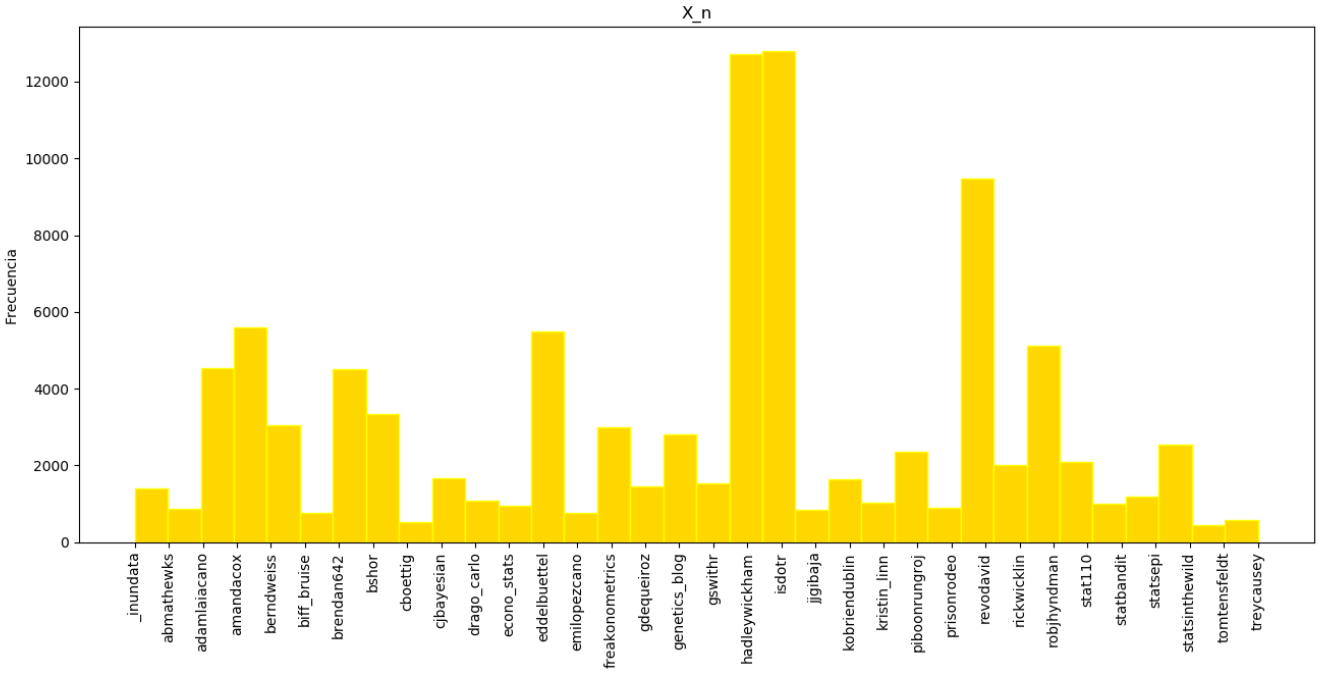

Figura 3: Histograma de frecuencia en que X_n tomó el valor $x \in S$ $(N(i,n))$, con S el espacio de estados tal que S =Usuarios.

Cabe destacar que se realizó varias veces la simulación y el histograma siempre asumía una forma parecida a la obtenida en la figura 3, lo cual es evidencia de que posiblemente existe una distribución invariante para este proceso.

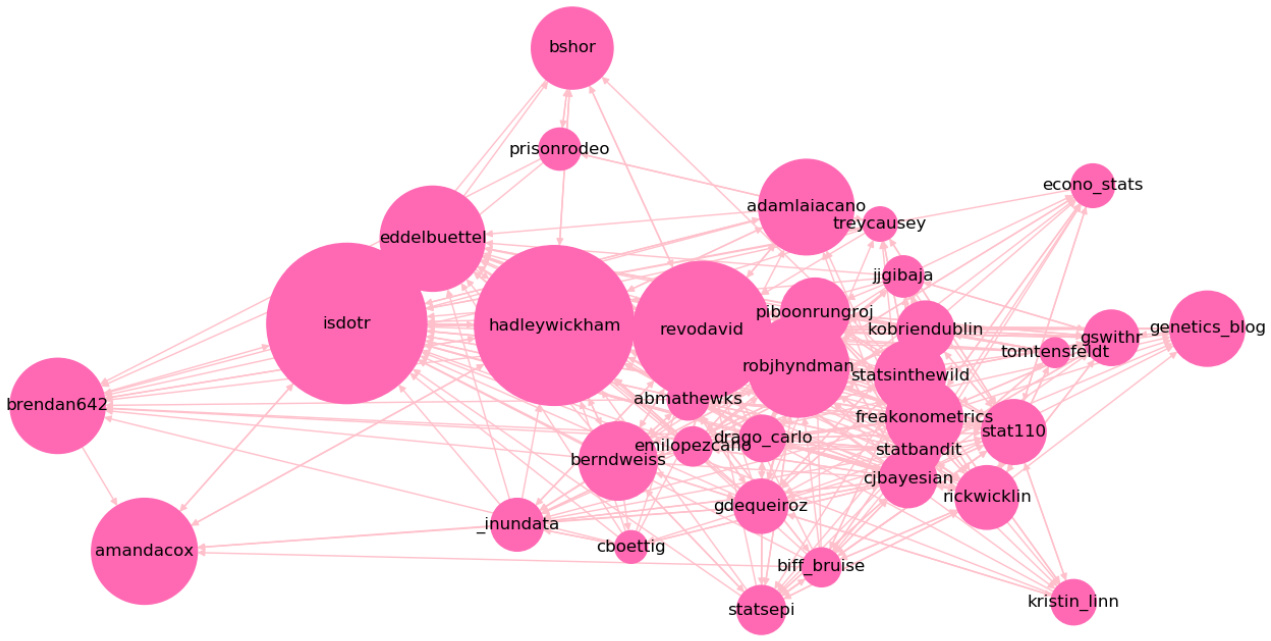

Figura 4: Gráfica con el tamaño del nodo i proporcional a $N(i,n)$.

i. **¿Cuáles son las fortalezas y cuáles las debilidades del algoritmo? (matemáticas, computacionales y de aplicación al mundo real).**
El algortimo PageRank calcula la importancia de una página de acuerdo a la siguiente

fórmula[1]:

$$PR(u) = (1 - d) + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Donde:

- B_u es el conjunto de páginas que tienen un enlace a la página u.
- $PR(u)$ es el PageRank de la página u.
- d es un factor de amortiguamiento tal que $d \in [0, 1]$.
- $PR(v)$ son los factores PageRank que tienen cada una de las páginas $v \in B_u$.
- $L(v)$ es el número total de enlaces salientes de la página v (sean o no hacia u).

Dada la simpleza de la fórmula, la única debilidad que podría tener es que por lo general se implementa para una cantidad exorbitante de datos (páginas), por lo que a pesar de su simpleza podría tardar mucho en hacer todos los cálculos.

ii. **¿Cómo podría manipularse el algortimo, como sitio (o sitios) interesado(s) en tener un alto PageRank?**

Se podría postear en las páginas más importantes (posibles) enlaces a nuestra página, para que así el algoritmo tome en cuenta esos enlaces, aun si la página misma no fue quien los puso.

iii. **Como diseñador del algoritmo interesado en la justicia, ¿cómo podría combatirse la manipulación anterior?**

Se podría modificar el algoritmo de tal forma que solo cuente los enlaces que fueron puestos por la misma página, o que cuente cuántas personas dan click en dichos enlaces.

iv. **Mencione al menos dos variaciones factibles del algoritmo y sus mejoras o pérdidas respecto a este algoritmo.**

Una de ellas podría ser la segunda propuesta de la respuesta anterior. Sin embargo, tendría un alto costo computacional el implementar un contador de visitas a cada página.

Dicho esto, la otra podría ser simplemente contar el número de visitantes a cada página considerada, cada cierto intervalo de tiempo. Esto nos daría un resultado más acertado a la realidad sobre la importancia de una página y podría ser computacionalmente más barato.

[1] `https://en.wikipedia.org/wiki/PageRank`