UNIVERSIDAD ADOLFO IBAÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS

Tarea 1

ING560: Algebra Lineal y Optimización para Data Science

Profesor: Miguel Romero

Fecha de Entrega: 05 de Noviembre 2020

Indicaciones:

1. Se aceptarán tareas enviadas después de la fecha de entrega. Se descontará 1.0 punto sobre la nota final por cada día de retraso.

2. La tarea se puede hacer en grupos de a lo más 2 personas.

3. Debe entregar un **informe** con sus respuestas a los problemas. Junto con la tarea se adjunta un archivo `main.py` el cual deberá completar para implementar los algoritmos pedidos. En las preguntas computacionales (Problema 2, pregunta 2) no basta con entregar el código, sino que **debe** explicar brevemente la idea de su implementación. Para la preparación del informe se recomienda el uso de LaTeX.

4. Puede bajar las últimas versiones de `Python`, `Numpy` y `Scipy` desde https://www.python.org/, https://numpy.org/ y https://www.scipy.org/.

Problema 1 (30%)

En clases vimos cómo aplicar PageRank a un grafo dirigido. Recuerde que en un grafo dirigido tenemos un conjunto de nodos y algunos links entre nodos. Una forma común de implementar el algoritmo de PageRank es a través del siguiente algoritmo iterativo. Comenzamos por asignar el PageRank de cada nodo $i \in \{1, \dots, n\}$ como $PR_0(i) = 1/n$. En otras palabras, al comienzo todos los nodos tienen el mismo PageRank. Escribimos PR_0 para indicar que esto es el PageRank en el tiempo 0. Luego hacemos iteraciones. En cada iteración $t \geq 1$, el PageRank del nodo i se actualiza de la siguiente forma:

$$PR_t(i) = \frac{1-d}{n} + d * \sum_{j \in \text{in-nb}(i)} \frac{PR_{t-1}(j)}{\text{out-deg}(j)} \tag{1}$$

donde in-nd(i) es el conjunto de nodos j que apuntan a i, es decir, tal que existe un link de j a i, y out-deg(j) es la cantidad de nodos k a los cuales apunta j, es decir, tal que existe un link de j a k. Nuevamente ocupamos PR_t y PR_{t-1} para indicar el PageRank en el tiempo t y $t-1$, respectivamente.

Después de cada iteración t comparamos los valores del PageRank en el tiempo t y $t-1$, revisando si la siguiente condición se cumple:

$$\sqrt{\sum_{i=1}^{n} (PR_t(i) - PR_{t-1}(i))^2} \leq \varepsilon \tag{2}$$

donde ε es un valor pequeño que se le da como argumento al algoritmo (por ejemplo $\varepsilon = 0.0001$). Si la condición (2) se cumple, entonces el algoritmo se detiene y retorna como PageRank los valores en PR_t. De lo contrario, seguimos con la siguiente iteración.

1. (1.5 pts) Explique a qué corresponde la actualización de PR_t en la ecuación (1) en términos de matrices y vectores.

2. (1.5 pts) ¿A qué corresponde PR_t en términos de Cadenas de Markov?

3. (1.5 pts) Explique a qué corresponde la condición de PR_t y PR_{t-1} en la ecuación (2) en términos de vectores.

4. (1.5 pts) Explique entonces qué es lo que está haciendo el algoritmo y cómo se relaciona con la definición de PageRank (vista en clases).

Problema 2 (70%)

En clases vimos cómo calcular PageRank de un grafo dirigido a través de vectores y valores propios. Dado un grafo dirigido G con n nodos enumerados $\{1, \ldots, n\}$, la idea era definir una cadena de Markov cuya matriz de transición P tiene en la fila i y columna j la entrada p_{ij} dada por:

$$p_{ij} = \begin{cases} \frac{(1-d)}{n} + d \cdot \frac{1}{\text{out-deg}(j)} & \text{si hay un link de } j \text{ a } i \text{ en } G \\ \frac{(1-d)}{n} & \text{si no} \end{cases}$$

Luego gracias al Teorema de Perron-Frobenius siempre existía un vector propio $\mathbf{x} \in \mathbb{R}^n$ asociado al valor propio $\lambda = 1$ el cual tenía todas sus coordenadas ≥ 0 y la suma de sus coordenadas es $= 1$ (es decir, $\mathbf{x}$ representa una distribución de probabilidad sobre los nodos). El PageRank era precisamente este vector $\mathbf{x}$.

1. (0.75 pts) Suponga que ahora tenemos un grafo dirigido en donde cada link de j a i tiene un peso $w_{ij} \geq 0$. ¿Cómo modificaría PageRank, en particular, cómo definiría la matriz de transición P, para abordar este caso? Observe que el caso de grafos dirigidos sin pesos corresponde al caso particular en donde todos los pesos son $= 1$.

2. (4.0 pts) Utilizando las librerías `NumPy` y `SciPy`, y en particular, la función `eig` para obtener los valores y vectores propios de una matriz, implemente una función `PageRank(G, d=0.85)` la cual tiene como parámetro un grafo dirigido con pesos y el *dumping factor* d (cuyo valor por defecto es $= 0.85$). El grafo dirigido G es entregado como un objeto del tipo `ndarray` con entradas de tipo `float` y dimensión 2, el cual representa una matriz de $n \times n$, donde n es la cantidad de nodos, y la entrada en la fila i y columna j es el peso w_{ij} del link de j a i [1]. La salida de la función debe ser un objeto del tipo `ndarray` con entradas de tipo `float` y dimensión 1, el cual contiene el PageRank de los nodos de G[2].

   Ejemplos de uso:

```
# Grafo dirigido visto en clases (clase 14, slide 20)

import numpy as np
import scipy.linalg as la


G = np.array([
[0, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 1, 0, 0, 0, 0],
```

[1] Ojo: esta representación es la *transpuesta* de la definición típica de *matriz de adjacencia* de un grafo dirigido.

[2] Si bien, en principio, con este método obtenemos el PageRank teórico exacto, construir explícitamente la matriz de transición y trabajar con la (transpuesta) de la matriz de adjacencia a veces no es muy eficiente para grafos grandes. En ese caso, es común representar el grafo con *listas de adyacencia* (almacenar para cada nodo sólo sus vecinos) y utilizar la aproximación dada por el método iterativo descrito en el Problema 1.

```
[0, 1, 0, 0, 1, 1, 1, 1],
[0, 0, 0, 0, 1, 0, 1, 1],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 1, 0, 0, 0]
], dtype=float)

print(PageRank(G))

>>> [0.05141623 0.07686172 0.37997977 0.0508515 0.03315793 0.34877886 0.01875
0.04020399]


print(PageRank(G, d=0.1)) # Disminuimos el dumping factor a d = 0.1

>>> [0.11893137 0.12862734 0.14646382 0.12702598 0.1167342 0.13006474 0.1125
0.11965255]


# Grafo dirigido visto en clases (clase 14, slide 20) con pesos modificados.
Se disminuye el peso de los arcos hacia el nodo 3 para que deje de ser el nodo
más importante


H = np.array([
[0, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 1, 0, 0, 0, 0],
[0, 0.001, 0, 0, 0.001, 0.001, 0.001, 0.001],
[0, 0, 0, 0, 1, 0, 1, 1],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 1, 0, 0, 0]
], dtype=float)

print(PageRank(H))

>>> [0.21012051 0.22536691 0.16655023 0.09887465 0.04676448 0.17356322 0.01875
0.06001001]
```

3. (0.5 pts) ¿Qué obtenemos cuando llamamos a nuestra función `PageRank` con $d = 0$? ¿Por qué obtenemos eso?

4. (0.75 pts) Aplique su función `PageRank` (con el dumping factor por defecto $d = 0.85$) a los 2 datasets entregados `email-Eu-core-temporal-norm.txt` (986 nodos y 24929 links) y `wiki-Vote-norm.txt` (7115 nodos y 103689 links) y encuentre en cada caso el nodo con mayor PageRank. La información de estos 2 datasets la puede encontrar en

   http://snap.stanford.edu/data/email-Eu-core-temporal.html
   y http://snap.stanford.edu/data/wiki-Vote.html

   respectivamente. Los dos archivos entregados contienen versiones normalizadas de estos datos de manera que los nodos están enumerados de 1 a n, donde n es la cantidad de

nodos. Para cargar estos datos utilice la función `fromFiletoMatrix(file, num_nodes)` dada en `main.py` que recibe el nombre del archivo y la cantidad de nodos, y retorna la matriz que representa al grafo (la cual se le puede pasar a la función `PageRank`).