Métodos Computacionales
Primer Trabajo Práctico
Primer Semestre 2024

Introducción

En el año 1998, Sergey Brin y Lawrence Page presentaron el famoso paper *The anatomy of a large-scale hypertextual Web search engine* como doctorandos en la Universidad de Stanford, California. Este trabajo describe el algoritmo *PageRank*, que llevo a Page y Brin a la creación de la empresa norteamericana *Google*. El algoritmo provee un método sencillo y eficiente para poder ordenar las páginas web según su importancia o relevancia, utilizado por el motor de *Google* a la hora de mostrar resultados de una búsqueda.

Una forma simple de determinar la importancia de un paper científico (o una página web) es contar la cantidad de citas que tiene, lo cual permite hablar acerca del impacto que tuvo en trabajos posteriores y en la comunidad científica en general. Sin embargo, rápidamente nos podemos dar cuenta que la cantidad de citas no alcanza por si sola[1]. La idea central de PageRank es que no importa únicamente la cantidad de citas que tiene un trabajo, sino la calidad de las mismas. En este trabajo práctico implementaremos el algoritmo de *PageRank* y lo aplicaremos sobre una base de datos de literatura científica para poder ordenar trabajos en función de su impacto.

Método

Para modelar el problema, imaginemos un estudiante que lee trabajos científicos. Este estudiante, al leer un paper decide aleatoriamente con probablidad d continuar leyendo alguno de los trabajos citados; o con probabilidad $(1-d)$ proseguir leyendo cualquier otro paper disponible. En el primer caso, elige alguna de las citas de forma equiprobable. Si $p_1 \dots p_N$ son los papers disponibles para leer, y pensamos los trabajos leídos como una secuencia $X_1, X_2, \dots$ podemos escribir:

$$P(\text{leer el paper } p_i \text{ luego de leer paper } p_j) = P(X_{t+1} = p_i \mid X_t = p_j). \tag{1}$$

Y la formula que sigue esta probabilidad es:

$$P(X_{t+1} = p_i \mid X_t = p_j) = \begin{cases} \frac{d}{c_j} + \frac{1-d}{N} & \text{si } p_j \text{ cita a } p_i \\ \frac{1-d}{N} & \text{sino} \end{cases}, \tag{2}$$

donde c_j es la cantidad de trabajos citados por el paper p_j. Ahora bien, nos interesa saber para cada paper p_i la probabilidad que el estudiante lo este leyendo a tiempo $t+1$. Siguiendo la formula de las probabilidades totales, sabemos que:

$$P(X_{t+1} = p_i) = \sum_{j=1}^{N} P(X_{t+1} = p_i \ , \ X_t = p_j) \tag{3}$$

$$= \sum_{j=1}^{N} P(X_{t+1} = p_i \mid X_t = p_j) P(X_t = p_j) \tag{4}$$

$$= \frac{1-d}{N} + \sum_{j \in C(i)} \frac{d}{c_j} P(X_t = p_j), \tag{5}$$

siendo $C(i)$ el conjunto de índices de los papers que citan a p_i, es decir j está en $C(i)$ si p_j cita a p_i.

[1] Por ejemplo, utilizando este criterio, el trabajo de Watson y Crick acerca del descubrimiento del ADN "*Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid*" es del orden de diez veces menos relevante que el paper del 2016: "*Deep Residual Learning for Image Recognition*" (18k citas vs 211k citas respectivamente), un trabajo que propuso una arquitectura de red neuronal que tiene poco uso, tan solo ocho años después de su publicación.

Si consideramos $\mathbf{p}_{t+1} \in \mathbb{R}^N$ el vector:

$$\mathbf{p}_{t+1} = \begin{pmatrix} P(X_{t+1} = p_1) \\ P(X_{t+1} = p_2) \\ \vdots \\ P(X_{t+1} = p_N) \end{pmatrix},$$

a la matriz $W \in \mathbb{R}^{N \times N}$ tal que $W_{ij} = 1$ si p_j cita a p_i y $W_{ij} = 0$ sino. A la matriz diagonal $D \in \mathbb{R}^{N \times N}$ con $d_{ii} = \frac{1}{c_i}$. Y siendo $\mathbb{1}$ el vector de $\mathbb{R}^N$ de todos unos. Podemos escribir el resultado de la Ecuación 5 de forma matricial:

$$\mathbf{p}_{t+1} = \frac{1-d}{N}\mathbb{1} + d\,W D\,\mathbf{p}_t. \tag{6}$$

De esta forma, luego de muchos pasos (es decir a medida que t crece) aquellos papers que el estudiante tenga más probabilidad de estar leyendo son los que consideraremos de mayor impacto o relevancia. Intuitivamente, para que un paper tenga una probabilidad alta a tiempo t, los papers que lo citan también deben tener probabilidad alta, y los que citan a estos últimos, y así. La teoría de las probabilidades y las cadenas de Markov nos garantizan que a medida que t se va a infinito, las probabilidades van a converger a lo que llamamos una distribución estacionaria.

Implementación

Las ventajas del algoritmo descrito no son únicamente teóricas. En la práctica, utilizando las técnicas apropiadas, se logran algoritmos muy eficientes tanto en términos de complejidad computacional como de utilización de memoria. La principal optimización que haremos en este TP, será la utilización de matrices ralas.

Cada paper realiza como máximo unas 50 referencias. Por ejemplo, si estamos trabajando con una base de datos de 1000 papers, cada columna de la matriz W tendrá a lo sumo 50 elementos de valor unitario, y todo los restantes elementos serán nulos. Por lo tanto, tiene poco sentido almacenar la matriz en memoria explícitamente. Existen múltiples estructuras de datos para representar las matrices ralas, la más común es la lista de listas enlazadas (LIL), donde se almacenan únicamente las filas que tengan algún elemento no nulo, y de estas se almacena únicamente las columnas que cumplan la misma propiedad. Utilizando listas enlazadas para poder mantener en orden las columnas durante inserción de nuevos elementos y para poder recorrer velozmente cada fila en orden (especialmente necesario para suma y producto de matrices).

Típicamente para mayor eficiencia se implementa el siguiente esquema:

- Se almacena cada fila f como una lista enlazada, donde el valor de cada nodo es un par (c, n). c indica el número de columna, y n el numero de la matriz en la posición f, c.
- Para acceder velozmente a las filas, se las indexa a través de un diccionario. En este, la clave de cada fila es su índice y el valor almacenado es la lista enlazada.

Es sumamente importante que las listas enlazadas de cada fila estén ordenadas por índice. ¡Hay que tener especial cuidado a la hora de poblar la matriz!

Ejercicios

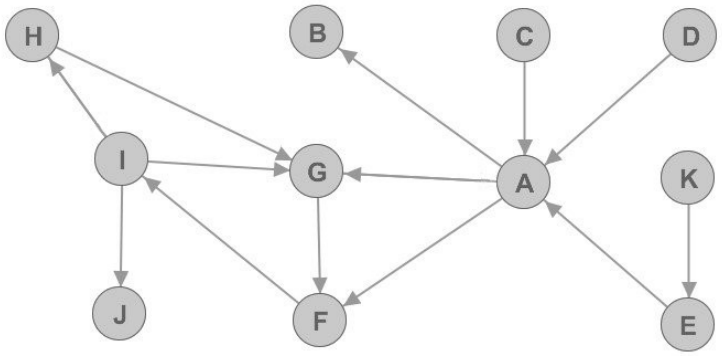
Ejercicio 1. Implementar en Python la clase MatrizRala, esta debe proveer métodos para:

- Indexación
- Suma
- Resta
- Multiplicación por un escalar
- Producto matricial

En el archivo *matrizRala.py* se encuentra el esqueleto de las funciones para que puedan completar. Además se provee la clase *ListaEnlazada*, con las típicas propiedades de esta estructura de datos. En el archivo *tests.py* se proveen tests (incompletos) para que evalúen sus implementaciones.

Ejercicio 2. Implementar el método de Gauss-Jordan para matrices ralas.
En el archivo *tests.py* se proveen tests (incompletos) para que evalúen sus implementaciones.

Ejercicio 3. Se tienen las relaciones de la siguiente figura:



Donde cada nodo $\{A, \ldots, K\}$ representa un paper y las flechas entre papers son las citas. Por ejemplo, la flecha entre papers A y F indica que el paper A ha citado al paper F.

- Crear las matrices W y D para la Figura 3 utilizando la clase MatrizRala que crearon.
- Si el sistema converge, encontraremos un vector de probabilidades $\mathbf{p}^*$ tal que:

$$\mathbf{p}^* = \frac{1-d}{N}\mathbb{1} + d\,W D\,\mathbf{p}^*.$$

  Expresar la ecuación como un sistema lineal ($A\mathbf{x} = \mathbf{b}$) y resolver para $\mathbf{p}^*$ utilizando $d = 0,85$. Comparar el resultado obtenido con el método iterativo utilizando una distribución equiprobable para la probabilidad inicial. Para realizar la comparación, graficar la diferencia absoluta entre $\mathbf{p}_t$ y $\mathbf{p}^*$ para valores de t hasta ver que converge.

Ejercicio 4. El archivo *papers.zip* posee dos tablas de una base de datos. La primera, *papers.csv* contiene un conjunto de trabajos científicos, con un campo para el *Id* que los identifica, un campo para su título, para sus autores, y para el año en que fue publicado. Por ejemplo:

```
96319,Helly-type theorems and geometric transversals,Rephael Wenger,1997
```

La segunda, *citas.csv* contiene dos columnas con los *Id*s del paper citate y el citado. Por ejemplo si el paper 17 cita al paper 96319:

```
17,96319
```

- Generar las matrices W y D a partir de la tabla *citas.csv*. Utilizar matrices ralas.
- Debido al tamaño del conjunto de datos, no podemos resolver explícitamente usando Gauss-Jordan el sistema. Utilizando $d = 0{,}85$ y asumiendo una distribución equiprobable para el tiempo 1, computar $\mathbf{p}_t$ hasta que la serie converja. Es decir, que la diferencia $\|\mathbf{p}_{t+1} - \mathbf{p}_t\|$ sea menor a cierto $\epsilon > 0$.
- Armar la lista de los 10 papers con mayor impacto. Comparar los resultados con aquellos obtenidos utilizando únicamente la cantidad de citas. ¿Qué conclusiones se pueden sacar?, ¿Qué algoritmo parece dar resultados más satisfactorios?

Condiciones de Entrega

Se deberán implementar todas las funciones sin usar librerías. Solo pueden Pandas para cargar el dataset. La entrega del trabajo debe consistir en los siguientes archivos:

1. `informe.pdf:` un informe o reporte en el que se describan las tareas realizadas, cómo fueron resueltas, los resultados de sus experimentos y conclusiones. Puede ser escrito en programas del tipo Word, Markdown o LaTeX, controlando que al exportar el archivo no haya errores de fórmulas ni de formato.
2. `matrizRala.py:` El archivo modificado con las implementaciones de los ejercicios 1 y 2.
3. `ejercicio_{3}.ipynb` y `ejercicio_{4}.ipynb:` un *Notebook* de `Jupyter` para los últimos dos ejercicios que contenga todo el código utilizado. El código debe ejecutar sin errores, es decir, debe ser reproducible sin la necesidad de cambios adicionales.

Además, se deben tener en cuenta las siguientes consideraciones:

- El trabajo debe realizarse en grupos de 2 alumnos.
- La fecha límite de entrega es el **domingo 12 de mayo**.
- La entrega debe ser realizada a través del campus por sólo 1 de los integrantes del grupo (no entregar por duplicado).
- En cada uno de los archivos entregados deben especificarse, arriba de todo, los nombres y apellidos completos de los integrantes del grupo.
- Las consignas sirven como guía de trabajo, se valorará el uso de técnicas vistas en clase como operaciones de vectores, matrices, y demás.
- Mantener el mayor nivel de prolijidad posible, tanto en los desarrollos como en el código. Las resoluciones deben estar escritas de forma clara y precisa.