Modelos de Interacciones Sociales: Taller 2

Facultad de Economía, Universidad de los Andes.

Octubre 13 de 2022

Instrucciones del trabajo:

El presente taller busca que se apliquen los conceptos vistos durante las clases Magistral y Complementaria de las semanas 4-8 del curso. Adicional a esto, con los ejercicios propuestos se busca desarrollar una mejor comprensión del uso de Python, de los paquetes vistos en clase y fomentar un razonamiento lógico que lleve a los estudiantes a comprender e implementar de una buena manera los tópicos tratados.

El taller se debe desarrollar en grupos de 4 personas (sin excepciones). Las respuestas al mismo deben ser entregadas en formato pdf, con el patrón de marcado: Cod1_Cod2_Cod3_Cod4.pdf. Por otro lado, se debe adjuntar el archivo .py que contenga las implementaciones correspondientes. Este último archivo debe venir marcado con el formato: Cod1_Cod2_Cod3_Cod4.py. En esta ocasión solo se recibirán archivos .py que tengan la estructura de librería, i.e. consten únicamente de paquetes cargados y las funciones que ustedes construyan, las cuales deben tener la estructura solicitada (inputs y outputs en la forma y orden que se piden). También se pueden incluir funciones auxiliares que ayuden a la implementación de las funciones principales. La fecha de entrega es el **01 de Noviembre de 2022 a las 23:59** a través del enlace publicado en Bloque Neón. Los archivos .pdf y .py deben estar en una carpeta comprimida (.zip) marcada así: T2_MIS_Cod1_Cod2_Cod3_Cod4.zip.

Con respecto a los factores que pueden afectar la nota se encuentran: (1) los códigos que no estén ordenados, debidamente comentados, marcados según el formato especificado o sin los nombres solicitados para los módulos y funciones, serán penalizados con 0.5 unidades por cada fallo; (2) aquellos scripts o fragmentos de script que no corran o no ejecuten la rutina solicitada anulan los puntos obtenidos por respuesta correcta; (3) los archivos .pdf que no sean legibles, no estén ordenados o no estén marcados con el formato solicitado tendrán la misma penalización que en el punto (1); (4) se dará un bono de 0.5 unidades a aquellos archivos .pdf creados con LaTeX. Como último elemento de estas instrucciones, la copia o plagio en este trabajo está totalmente prohibida e incurrir en esta práctica conlleva a una nota de cero (0) en el taller, así como a

las sanciones correspondientes tenidas en cuenta en el reglamento de estudiantes de la universidad.

Finalmente, los scripts deben estar organizados por puntos y subpuntos. Para ser consistentes con la organización, cada implementación, i.e. cada subpunto, debe contener una sección de funciones relevantes y posterior a esta la sección de implementación en la cual se ejecutan las funciones creadas para dar respuesta a lo solicitado. En este taller, los puntos que no sean demostraciones serán testeados y si las soluciones no son las esperadas se pondrá una nota de cero.

Puntos a Desarrollar:

0.1 Validación de teoremas y el Modelo de De Groot *(3.5 puntos.)*

En este punto se realizarán programas que permitan realizar una verificación del teorema de convergencia en cadenas de Markov, en el marco del modelo de De Groot. Así, se implementarán una serie de funciones que permitan realizar realizar la verificación sobre un grafo dado, de tamaño inferior a 100. Como equipo deben escribir funciones novedosas, por lo que deben crear sus propias funciones (no pueden usar las que están en los Notebooks) u optimizar y/o mejorar las funciones que ya se propusieron (las optimizaciones y/o mejoras deben salir de sus propias ideas y NO vale argumentar que estas son cambios en los objetos usados, reducción de espacios que ocupa el programa, etc.). Para este punto pueden usar funciones de NetworkX del tipo: has_path, successors, predecessors, etc. Así, realice implementaciones para los siguientes numerales:

1. Función que tome una matriz y verifique si es estocástica por filas o no. Así, la función debe ser matriz_ef(matriz:np_matrix) → bool, donde se testea si "matriz" es una matriz estocástica por filas y se retorna "True" si lo es. En caso contrario, se retorna "False".

2. Función que tome un listado de números, sin repeticiones, y obtenga TODOS los subconjuntos que se pueden armar de este listado, de un tamaño dado. Así, la función debe ser list_subsets(listado:list,length:int) → list, donde se recibe una lista "listado" y se retorna una lista que contiene en su interior listados que representan cada uno de los subconjuntos de "listado" de tamaño "length". Se permite el uso del paquete itertools.

3. Función que calcula el máximo común divisor de un listado de enteros positivos. Así, la función debe ser mcd(listado:list) → int, donde se retorna el máximo común divisor de los números que están dentro de la lista "listado".

4. Función que diga si un grafo es aperiódico. Así, la función debe ser grafo_aperiodico(grafo,conj_nodes) → bool, donde se retorna "True" si el

grafo inducido por "grafo" y "conj_nodes" es aperiódico y "False" de lo contrario.

5. Función que diga si un grafo es fuertemente conexo. Así, la función debe ser grafo_fc(grafo,conj_nodes) $\rightarrow$ bool, donde se retorna "True" si el grafo inducido por "grafo" y "conj_nodes" es fuertemente conexo y "False" de lo contrario.

6. Función que diga si en un grafo dado, los nodos en este conforman un conjunto cerrado. Así, la función debe ser grafo_cc(grafo,conj_nodes) $\rightarrow$ bool, donde se retorna "True" si el conjunto de nodos "conj_nodes" conforma un conjunto cerrado dentro del grafo "grafo" y "False" de lo contrario.

7. Función que tome un grafo con pesos y revise si este representa una cadena de Markov convergente. Se deben realizar las siguientes tareas (en orden):

   (a) Revise si la matriz de adyacencia es estocástica por filas.

   (b) Si el item anterior se cumple, entonces se debe construir una nueva matriz de adyacencia que conserve las entradas iniciales si son cero y tenga 1's donde las entradas de la matriz original sean mayores a 0. Con esta, generar un nuevo grafo (en general el grafo es dirigido).

   (c) Sobre el nuevo grafo, testear si se cumple la proposición: $\forall S \subseteq N$: (S es fuertemente conexo $\wedge S$ es cerrado) $\rightarrow$ (S es aperiódico). Recuerde usar las reglas lógicas que dicen que $\neg(P \rightarrow Q) \equiv P \wedge \neg Q$; $P \rightarrow Q \equiv \neg P \vee Q$ y la regla que dice que si P es falso, entonces $P \rightarrow Q$ es cierto. Recuerde que si una proposición A es verdadera (falsa), entonces "No A", i.e. $\neg A$, es falsa (verdadera). Como última ayuda de este numeral, recuerde que un subconjunto de nodos inmersos en una estructura de red genera, potencialmente, un subgrafo. Tenga en cuenta esta idea para testear si un subconjunto de nodos es aperiódico y fuertemente conexo, i.e. cuando usted introduzca el grafo general en su función y el subconjunto de nodos relevante (inputs), genere un nuevo subgrafo a partir de estas dos cosas y trabaje sobre este para estas dos funciones en particular. En cuanto a la función de Conjunto cerrado de nodos, NO induzca el subgrafo, pues ello conlleva a una implementación incorrecta.

   La función debe construirse así, g_convergente(grafo) $\rightarrow$ bool. Así, se debe retornar "True" si la matriz de adyacencia del grafo "grafo" es convergente y "False" en caso contrario. Recuerde que esta función emplea TODAS las funciones creadas en los literales anteriores.

0.2 Análisis básico de texto *(3.5 puntos.)*

En este punto debe plantear funciones que le permitan realizar análisis de texto. Está permitido utilizar funciones de numpy que permitan manejar ciertas características de vectores u operaciones de manera más sencilla. Así, debe implementar lo siguiente:

1. Proponga una función que realice la limpieza de texto. Esta función debe tener la estructura limpieza_txt(listado:list)→list, donde "listado" es una lista de textos y se retorna una lista con los textos originales luego de hacerles la limpieza de texto.

2. Construya una función que le permita extraer un iterable que contenga las palabras que aparecen en los textos. La función debe tener la estructura palabras_txt(listado:list)→list, donde "listado" es una lista que sale del punto anterior y se retorna una lista con las palabras que están en los textos originales limpios (cada palabra debe aparecer una única vez).

3. Construya una función que le permita extraer un iterable que contenga las frases (no pueden existir repeticiones) que están contenidas en los textos. La función debe tener la estructura frases_txt(listado:list)→list, donde "listado" es una lista que sale del punto 1 y se retorna una lista con las frases presentes en los textos.

4. Construya una función que le permita obtener los vectores de frecuencia (aparición de palabras en las frases), para cada una de las palabras. La función debe retornar un listado con estos vectores. La estructura de la función es la siguiente: frecuencia_txt(listaPalabras:list,listaFrases:list)→list, donde "listaPalabras" es el listado de palabras en los textos y "listaFrases" es el listado de frases en los textos. Se debe retornar un listado que contenga los vectores de frecuencias.

5. Construya una función que le permita calcular los "idf". La estructura de la función es idf_calc(listaPalabras:list,listaTextosLimpios:list)→list, donde "listaPalabras" es el listado que contiene las palabras y "ListaTextosLimpios" es la lista que contiene los textos limpios. Se retorna una lista con los idf calculados.

6. Construya una función que le permita calcular el idf-midified-cosine entre dos frases. La función debe tener la estructura idf_mod_cosine(frase1,frase2,idf)→float, donde "frase1" y "frase2" son los vectores de frecuencia de aparición de un par de frases e "idf" es el vector de idf que calcula la función anterior.

7. Construya una función que le permita calcular el coseno entre dos frases. Luego, debe sumarle 1 a este coseno y dividir el resultado por 2. Debe usar los vectores de frecuencia. La función debe tener la estructura aprox_cosine(frase1,frase2) → float.

8. Construya una función que le permita calcular el parecido de dos frases, usando los vectores de frecuencia, así: 1) Se normalicen las frases (los vectores), 2) se resten los vectores y se saque la norma de esta resta, 3) se divida el resultado anterior por 2, 4) se le reste el resultado anterior a 1. La función debe tener la estructura aprox_len(frase1,frase2) → float.

9. Construya una función que le permita, dados los vectores de frecuencia y un indicador de función de similaridad entre vectores (alguna de

las tres propuestas anteriormente), generar una matriz de pesos. Con esto, calcule el PageRank sobre esta matriz y retorne el vector resultado. Use la función "nx.pagerank()". La función debe tener la estructura lex_rank(vectoresFrecuencia,simFun) → np.vector, donde "vectoresFrecuencia" es un listado que contiene los vectores de frecuencia de cada una de las frases y "simFun" es un str indicador que contempla los siguientes casos:

- "idf" para correr la función idf_mod_cosine()
- "cos" para correr la función aprox_cosine()
- "len" para correr la función aprox_len()

Dado esto, la función debe crear la matriz de pesos según la función de similaridad que el usuario elija. Claramente, se debe retornar el vector de PageRank de la matriz resultante de la elección y este vector debe ser un vector columna de numpy.

0.3 Demostraciones *(3.0 puntos.)*

Sea (N,A) una red con conjunto de nodos $N = \{1, 2, 3, \ldots, m\}$ y matriz de adyacencia $A \in M_{mxm}(\{0, 1\})$. Note que en este caso no se asume que la red sea dirigida o no y claramente se tiene que $m \in \mathbb{Z} \wedge m > 1$. Sea también $f : \mathbb{R} \to \{0, 1\}$, dada por:

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Ahora, entienda bien la siguiente definición:

$$I_{i,j} = \{k \in \{1, ..., m-1\} | f((A^k)_{i,j}) = 1\}$$

Realice las siguientes demostraciones.

1. Muestre que el número de caminos de longitud k que van del nodo i al nodo j son $(A^k)_{i,j}$. Ayuda: Busque qué es una prueba por inducción y aplique este método de demostración a esto sobre k. Establezca el caso base en $k = 1$ y el paso inductivo para $k \leq m - 1$.

2. Muestre que la longitud del camino más corto que va de un nodo i a un nodo j es:

$$\Gamma_{i,j} = \begin{cases} \min(I_{i,j}) & , I_{i,j} \neq \emptyset \\ \infty & , I_{i,j} = \emptyset \end{cases}$$