Modelos de Interacciones Sociales: Taller 3

Facultad de Economía, Universidad de los Andes.

Noviembre 08 de 2022

Instrucciones del trabajo:

El presente taller busca que se apliquen los conceptos vistos durante las clases Magistral y Complementaria de las semanas 9-12 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 **25 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: T3_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 Betweenness y Girvan-Newman *(6.0 puntos.)*

A continuación se proponen los pasos y funciones para la implementación del algoritmo de Grivan-Newman para identificar comunidades.

1. Cree una función que le permita generar un árbol (note que en el sentido estricto lo que se pide obtener en Girvan-Newman no es un árbol, pero para efectos prácticos lo llamaremos así). La estructura de la función debe ser gn_p1(nodo:int,red:nx_DiGraph)→nx_DiGraph, donde se introduce una red (no necesariamente conexa) y un nodo que será el llamado nodo "raíz" del árbol a construir. El output de la función debe ser el árbol que se construye a partir de la red inicial tomando el nodo seleccionado como raíz. Si el nodo en cuestión NO está en la red, se debe retornar una excepción que diga: "Introduzca un nodo en la red".

2. Cree una función que le permita tomar un árbol y un nodo, el cuál será el llamado nodo "raíz". La función debe crear un atributo de nodos en el árbol que se llame "n_caminos" y llenarlo con el número de caminos más cortos que van de los nodos del árbol a la raíz. La función debe tener la estructura gn_p2(nodo:int,arbol:nx_DiGraph)→None, donde "arbol" es el árbol que eventualmente se recibirá del punto anterior y "nodo" es la raíz sobre la cuál se construye el mismo.

3. Cree una función que le permita asignar pesos a los nodos y enlaces según la tercera parte del algoritmo de Girvan-Newman. Para esto se debe proveer un árbol, su raíz y se debe generar un nuevo atributo de nodos en el árbol que se llame "pesos". Así mismo, para los enlaces debe generar un atributo que se llame "e_pesos", el cuál contendrá los pesos de los enlaces. La estructura de la función es gn_p3(nodo:int,arbol:nx_DiGraph)→None, donde "arbol" es el árbol que eventualmente se recibirá del punto anterior y "nodo" es la raíz sobre la cuál se construye el mismo (desde el paso 1).

4. Cree una función que tome una red y le cree un atributo de enlaces llamado "e_pesos". Habiendo hecho esto, la función debe correr los pasos

1-3 del algoritmo de Girvan-Newman para cada uno de los nodos dentro de la red y extraer los "e_pesos" del árbol resultante en cada paso para sumarlos al atributo de la red. La estructura de la función debe ser gn_peso_enlace(red:nx_DiGraph)→None, donde "red" es la red a la cuál se le aplicará el algoritmo de Girvan-Newman.

5. Cree una función que tome una red, un atributo de enlace y un número llamado "umbral". Esta debe generar una nueva red que se construye a partir de los enlaces que tienen un puntaje menor que el umbral (i.e. el atributo de enlace es menor estricto que el número "umbral"). Así, la estructura de la función debe ser gn_comunidades(red:nx_DiGraph,atributo:str,umbral:float)→nx_DiGraph, donde "red" es la red a aplicarle Girvan-Newman, "atributo" es el nombre del atributo de enlace con el que se van a crear las comunidades y "umbral" es el número que va a determinar cómo se crean las comunidades.

6. Cree una función que tome una red y corra el algoritmo de Girvan-Newman completo para un "umbral" dado. Esta debe recibir una red, aplicarle todos los pasos anteriores para el "umbral" establecido y retornar una red que evidencie las comunidades que se querían crear. Si lo quiere ver de alguna manera, en esta función debe introducir todas las anteriores. La estructura debe ser gn_definitivo(red:nx_DiGraph,umbral:float)→nx_DiGraph, donde "red" es la red en cuestión y "umbral" es el número que va a determinar cómo se crean las comunidades.

Aclaración: Puede generar atributos, conjuntos, listados y demás cosas que usted necesite en este punto, pero las funciones pueden utilizar únicamente los argumentos que se piden. Así, si usted genera listados, conjuntos, etc., no puede pasarlos como input a una función ni llamarlos desde la memoria de trabajo, pues se cuenta como argumento no válido y hace que su implementación sea incorrecta.

0.2 Clausura Triádica Fuerte y Puentes Locales *(4.0 puntos.)*

A continuación se proponen los pasos y funciones para la implementación de un algoritmo que permite realizar la prueba sobre redes particulares del teorema que relaciona la propiedad de Clausura Triádica Fuerte y la propiedad de ser Puente Local.

1. Proponga una función que permita revisar si un enlace es puente local o no. Para llevar a cabo esta tarea, debe tomar la red en cuestión y generar en ella un atributo de enlace llamado "puente". Posterior a ello, la función debe asignar True y False a cada uno de los enlaces en este atributo, según si el enlace es puente local (True) o no (False). La estructura de la función debe ser cp_puentes(red:nx_DiGraph)→None, donde "red" es la red en cuestión y a la que se le debe generar el atributo "puente" sobre sus enlaces.

2. Proponga una función que permita revisar si un nodo cumple la propiedad de la Clausura Triádica Fuerte. Para ello, usted recibirá una red con un atributo de enlaces que se llama "relacion". Acá, para un enlace en particular, "relacion" será True si este tiene la propiedad de ser Fuerte y "relación" será False en caso de ser débil. Así, usted debe tomar esta red (la cuál tiene el atributo de enlaces "relacion") y generar en esta un nuevo atributo de nodo que se llame "ctf". Este atributo debe ser llenado con False si el nodo incumple la propiedad de Clausura Triádica Fuerte y con True en caso contrario. La estructura de la función debe ser cp_ctf(red:nx_DiGraph)→None, donde "red" es la red en cuestión.

3. Proponga una función que reciba una red con tres atributos: uno de nodo llamado "ctf" y dos de enlace llamados "relacion" y "puente". Con esto, se debe verificar que los atributos que tiene la red en cuestión hayan sido construidos de acuerdo a las reglas que establecen las definiciones propuestas anteriormente, en particular la de Clausura Triádica Fuerte. Así, se verificará la construcción correcta de los atributos si cumple la siguiente afirmación: "Si un nodo en una red satisface la propiedad de clausura triádica fuerte y tiene por lo menos dos vecinos con enlaces fuertes, entonces en cualquier puente local en el que esté involucrado debe ser un enlace débil.". Si lo anterior se cumple la función debe retornar True y en caso contrario, False. La estructura de la función debe ser cp_revision(red:nx_DiGraph)→bool, donde "red" es la red con los atributos mencionados.

4. Proponga una función que reciba una red con un único atributo llamado "relación". A partir de esto y usando las dos primeras funciones de este punto, genere en esta red los nuevos atributos "ctf" y "puente". Luego corra sobre la red resultante la función cp_revision y retorne su resultado. Así, la estructura de la red es cp_construccion(red:nx_DiGraph)→bool, con "red" siendo la red que tiene el atributo "relacion" y el booleano que se retorna es el que sale de aplicar cp_revision a la red resultante luego de haber aplicado cp_puentes y cp_ctf.