Cápsula 1: d3-force

Hola, bienvenidxs a una cápsula del curso Visualización de Información.

En esta hablaré sobre "d3-force", un paquete de D3.js que permite realizar simulaciones de fuerza. Esto nos permitirá realizar procesos de posicionamiento dirigido por fuerzas para visualizar redes.

Comenzaremos por lo básico, que es cargar un *dataset* de red para usarse con "d3-force", y definir una simulación de fuerzas básica.

En este caso cargaremos un archivo JSON, ya que este nos permite suficiente flexibilidad para definir nodos y enlaces en el mismo archivo, siguiendo la misma estructura de objetos que esperaría D3. Podrás ver en pantalla que este archivo define dos campos separados: los nodos y los enlaces.

En el caso de los nodos, cada uno cuenta con dos atributos: "nombre" y "color". Por otro lado, los enlaces son objetos también, que mediante campos "source" y "target" referencian qué nodos conectan, mediante sus nombres. Uso estos nombres de campos ("source" y "target") porque son convención que usará D3.js para cargar dichos enlaces e identificar qué nodos se conectan.

Más campos se pueden agregar perfectamente a nodos y enlaces, o incluso podrían cargarse desde otro formato, como tabular. En casos de formatos distintos sería necesario en el mismo programa pre-procesar de forma que se generen objetos con una estructura similar a la mostrada, o específicamente, la estructura de los enlaces.

Además, también es importante que exista alguna forma de identificar cada nodo del resto, y así usar dicha llave para definir las relaciones de enlaces. En este caso el nombre funciona como llave por que así fue considerado en este caso.

En este programa de ejemplo cargaremos el *dataset*, sacaré los arreglos de nodos y enlaces en variables separadas, y llamaré a una función que iremos completando para producir nuestra visualización.

Esta función recibe los nodos y enlaces cargados, y hasta el momento tengo las definiciones mínimas para generar una simulación de fuerzas de D3. La primera sentencia produce el objeto de simulación mediante la función "d3.forceSimulation", que se encarga de toda la magia tras bambalinas de simular fuerzas. Esta función recibe los nodos a simular, que son los objetos que interactuarán mediante fuerzas.

Luego, mediante métodos “force” se definen y agregan distintas funciones que simulan fuerzas. Con esto es posible personalizar la simulación para que considere sólo fuerzas que le explicitamos.

Por ejemplo, “d3.forceLink” genera una fuerza de enlaces de red, y recibe los enlaces definidos. Veremos los detalles de distintas fuerzas más adelante, por ahora las trataremos como cajas negras.

Finalmente, mediante “on” se suscribe al objeto de simulación una función sobre un evento “tick”. Esto produce que en cada actualización de la simulación, se llame a la función entregada. ¡Y esto es suficiente para comenzar la simulación! Al definirla, esta comienza inmediatamente a funcionar, por lo que sí se llama a esta función completa, comenzará al llegar a este punto del programa.

Para que veamos ciertas cosas interesantes que produce generar la simulación, hago dos tipos de impresiones. Primero, imprimo justo después de la simulación el arreglo de enlaces que habíamos cargado originalmente. Por otro lado, en la función de actualización llamó a una función que imprimirá atributos de un nodo de cierto índice. Por defecto escogí el primero.

Si ejecutamos el programa y vemos la consola, veremos muchas impresiones. Partiremos revisando el arreglo de enlaces.

Como recordarán, los enlaces que cargamos en el archivo solo contaba con los nombres de los nodos que conectaban, pero aquí vemos que los campos “source” y ”target” tienen referencias a objetos completos. Y no cualquier objeto, los nodos que cargamos en el *dataset* originalmente de hecho. Este trabajo lo hizo la simulación, o específicamente, la fuerza de enlace que le agregamos.

Al mismo tiempo, podemos notar que estos objetos de nodos tienen más atributos de los que definimos al cargarlos. Estos son atributos generados por la simulación de fuerzas, donde los más importantes a notar son los campos “x” e “y”, que se refieren a coordenadas de los nodos para colocarlos en pantalla.

Y eso es lo que apreciamos en las siguientes impresiones, como evolucionan las coordenadas “x” e “y” en cada actualización de la simulación. Vemos que comienza con valores 312.316 y 297.62 y en cada impresiones siguientes estos valores cambian. ¡Se están simulando fuerzas entre nodos y sus posiciones simuladas van cambiando en los objetos que cargamos!

Lo único que nos faltaría sería ver los nodos en pantalla. Gracias a que la simulación agrega atributos de posiciones en nuestros nodos, y los enlaces tienen referencias a dichos nodos, la generación visual de estos elementos es muy fácil mediante *data join*.

Agregaremos entonces líneas por cada objeto de enlace en nuestro arreglo cargado y modificado, y agregaremos un círculo por cada nodo. Pintaremos cada nodo según el color

que tenía definido en el archivo base. ¡Pero aún no definiremos las posiciones de estos elementos!

Esto porque como las posiciones cambian constantemente, tiene más sentido generar la sentencia que cambia los valores posicionales de las marcas en la función de actualización de la simulación.

Así, a cada círculo se le indica que cambie sus posiciones según los valores en el dato de nodo que tenga asociado, y las líneas definen sus extremos en base a los objetos referenciados en sus campos “source” y “target”.

Si lo probamos, ¡vemos el resultado! Nodos, líneas que los conectan y que se mueven en el espacio disponible sin alejarse mucho entre ellos.

Ahora probaré el mismo programa con un *dataset* que tiene muchos más nodos y enlaces, y veremos cómo también funciona. Podemos también apreciar como la simulación efectivamente mantiene cerca nodos enlazados, y permite dar una idea de la topología general de la red.

Con eso termina el contenido de esta cápsula. Recuerda que si tienes preguntas, puedes dejarlas en los comentarios del video para responderlas en la sesión en vivo de esta temática. ¡Chao!