Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Tarea 2

IIC2026 – Visualización de información
Entrega: 11 de octubre de 2019, 19:59

Esta tarea pretende profundizar los conceptos de visualización y **D3.js** aprendidos durante las últimas semanas en clases y ayudantías. Esta tarea es **individual** y se entrega en su repositorio creado a partir de la siguiente **invitación**.

1. Visualización Árboles de Decisión [2.5 puntos]

1.1. Contexto

Los Árboles de Decisión son, debido a su sencillez y estructura auto explicativa, uno de los métodos de predicción más usados en varias áreas profesionales, desde *Machine Learning* hasta Economía. En esta tarea deberán ser capaces de visualizar dos modelos de Árbol de Decisión y un resumen de las clasificaciones realizadas por cada uno.

1.2. *Datasets*

Los datos a utilizar en esta tarea corresponden a la información de los pasajeros del Titanic[1] y los modelos fueron entrenados para predecir si dicho pasajero sobrevivía o no. Los *datasets* corresponden a archivos *JSON*, los cuales representan los modelos ya entrenados de Árbol de Decisión. En particular, el formato de estos corresponden a un diccionario con dos *keys*: *nodes* y *links*.

1.2.1. *nodes*

Esta *key* contiene una lista de diccionario. Cada diccionario representa un nodo del árbol y el formato del diccionario varía si es que el nodo es hoja o intermedio, y en caso de ser intermedio, si el dato utilizado para definir las reglas de división es categórico o numérico. No obstante, todos los nodos comparten las siguientes *keys*:

- *id*: identificador único del nodo. Es un número entero mayor o igual a 0.
- *is_leaf*: booleano que indica si el nodo es hoja. Puede ser `true` o `false`.

[1] Puede leer el enunciado mientras escucha esta **musica de fondo**.

Nodo hoja

En caso que el nodo sea hoja (*is_leaf* es `true`), el diccionario va a incluir las siguientes *keys*:

- *survival*: cantidad de personas que fueron clasificadas como sobrevivientes en este nodo.
- *not_survival*: cantidad de personas que fueron clasificadas como no sobrevivientes en este nodo.

Nodo intermedio

En caso que el nodo sea intermedio (*is_leaf* es `false`), el diccionario va a incluir las siguientes *keys*:

- *attribute*: nombre del atributo a ocupar para realizar la división.
- *is_categorical*: booleano que indica si el atributo utilizado para dividir es categórico. Puede ser `true` o `false`.
- *value*: valor utilizado por el nodo para realizar la división. En caso que el atributo sea numérico, *value* será un número de tipo `float`. En caso de ser categórico, *value* será una lista de `string` que representan las categorías a verificar.

1.2.2. *links*

Esta *key* contiene una lista de diccionario. Cada diccionario representa una arista del árbol. Considerar que el árbol es direccionado, es decir, el nodo X puede estar conectado con el Nodo Y, pero no lo contrario. Cada diccionario va a incluir las siguientes *keys*:

- *source*: ID del nodo a donde parte la arista.
- *target*: ID del nodo de donde llega la arista.
- *direction*: valor que indica si el nodo de *target* estará a la derecha (`right`) del nodo *source* o estará a la izquierda (`left`). Un nodo estará a la izquierda si el valor del atributo a verificar está en la lista (en caso de ser categórico) o es menor al valor (en caso de ser numérico). En otro caso estará a la derecha.

En resumen, el formato del *datasets* sería del siguiente modo:

```
{
    "nodes": [
        {
            "id": int,
            "is_leaf": bool,
            ...
        }, ...
    ],
    "links": [
        {"target": int, "source": int, "direction": string}, ...
    ]
}
```

1.2.3. Cargar datos

Para cargar los *datasets* con D3, deben utilizar los siguientes *links*:

- **Modelo 1**
- **Modelo 2**

Pueden ver un ejemplo de como cargar los JSON desde un *link* con la **ayudantía 6** del curso.

1.3. Características

Para cada uno de los *datasets* mencionados anteriormente, deberán implementar una visualización de un grafo cuya estructura corresponda a la del árbol de decisión descrito. Es importante que la visualización debe representar correctamente la estructura de un árbol, el cual puede ser horizontal o vertical, y que incluya la dirección de las aristas. No se aceptarán, por ejemplo, nodos hijos que estén representados con menor profundidad [2] que sus padres (Ver figuras 1 y 2). Son libres de utilizar D3, Javascript o un código inicial de Python para lograr este objetivo. Se evaluará que el árbol resultante no contenga aristas entrecruzadas.

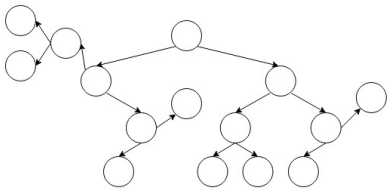


Figura 1: Árbol mal representado

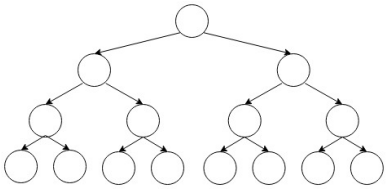


Figura 2: Árbol correctamente representado

Además, debe ser posible interactuar con su árbol, específicamente, se solicitan las siguientes implementaciones:

- `Hover Nodo intermedio`: Mostrar su información mediante un *tooltip*. En particular, mostrar el ID del nodo, nombre del atributo utilizado para generar la división, que tipo de atributo es (categórico o numérico) y el/los valores usados para generar la división. El formato del texto queda a su criterio.
- `Hover Nodo Hoja`: Para este tipo de nodo, se debe mostrar, mediante un *tooltip*, el ID del nodo y la cantidad de datos que fueron clasificados por clase. Además se deberá destacar la ruta de decisiones tomada a lo largo del árbol para llegar a ese nodo. Queda a su criterio la forma de destacar la ruta, pero se va a evaluar que las decisiones de diseño estén correctamente justificadas.

[2] Numero de aristas que hay que recorrer desde la raíz hasta el nodo

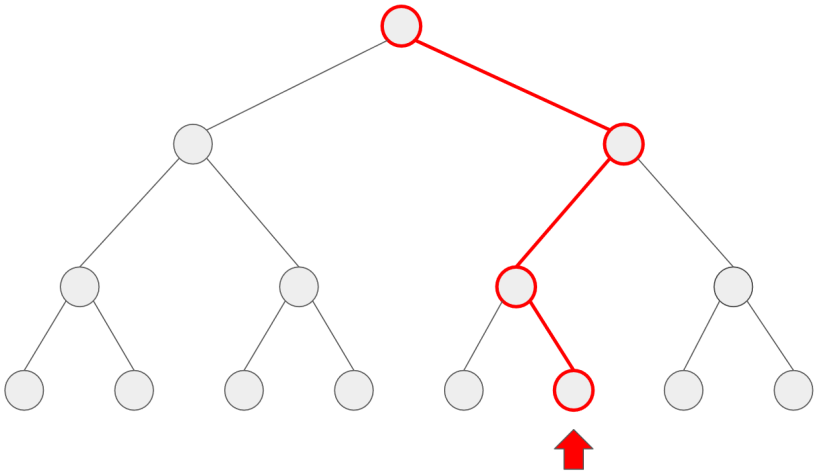


Figura 3: Ejemplo de ruta destacada

2. Vistas Coordinadas [2.5 puntos]

El idiom anterior permite visualizar el modelo, pero ahora queremos visualizar el resultado de aplicar este modelo a un *dataset*.Dado este motivo, deberán agregar a lo realizado anteriormente, un **Stacked Bar Chart** al lado de cada árbol, dónde el `EJE Y` del mismo corresponde a las dos clases posibles que clasifica el árbol, y el `EJE X` es la cantidad de datos que pertenecen a esa clase. Estos dos gráficos deben comenzar vacíos, y la forma de agregar y eliminar datos de estos es a través de dos interacciones con sus respectivos árboles y ambas interacciones deben hacer que los datos se carguen de forma fluida al **Stacked Bar Chart**, es decir, utilizar **transition**. Las dos interacciones a implementar son:

- `Click Nodo Hoja`: Se agrega al *stack* la cantidad de datos que se clasificó para cada clase en ese nodo.
- `Click en nodo previamente agregado`: Se retira del *stack* la información relacionada a ese nodo.

A continuación se presentan dos ejemplos de *layouts* posibles con el efecto de hacer *click* en un nodo.

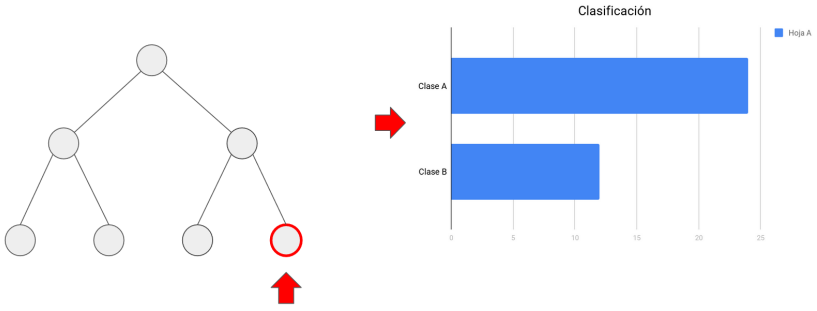


Figura 4: Interacción al hacer click en una hoja

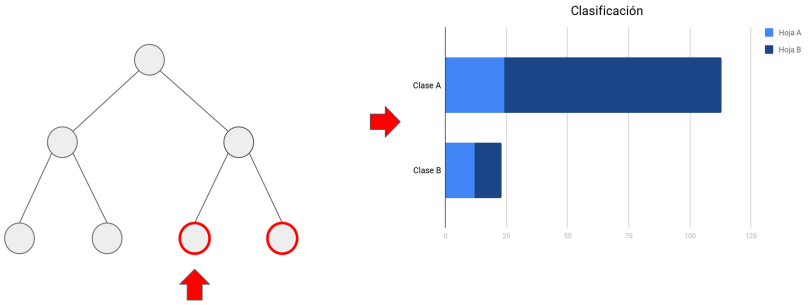


Figura 5: Interacción al hacer click en dos hojas

3. Bonus: Predicción [0.5 puntos]

Si nosotros hubiéramos subido al Titanic, ¿hubiéramos sobrevivido?. Ahora es el momento de saberlo con el siguiente bonus. Además de lo ya implementado, pueden agregar un formulario **con HTML o Javascript** que solicite todos los parámetros del modelo[3], y que tenga un botón "`Predecir`", el cual deberá predecir si el pasajero con dicha información sobrevive o no. Para determinar si sobrevive, puede considerar las siguientes dos opciones:

- Cuando llegue al nodo hoja, si la clase *survival* posee mayor cantidad de datos clasificados que *not_survival*, entonces se responderá *survival*. En otro caso retorna *not_survival*. Por ejemplo, si un nodo tiene 50 datos clasificados como *not_survival* y 10 como *survival*, entonces la respuesta será *not_survival*.
- Cuando llegue al nodo hoja, se genera una respuesta de forma aleatoria. Esta aleatoriedad se hará con una probabilidad por clase según la cantidad de datos clasificados por clase dividido la cantidad total de datos clasificados en dicho nodo. Por ejemplo, si un nodo tiene 50 datos clasificados como *not_survival* y 10 como *survival*, entonces la respuesta será generada con un 50/60 % de probabilidad de ser *not_survival* y 10/60 % de ser *survival*.

Además de esto, deberá destacar tanto la ruta que se siguió en cada modelo con el nodo hoja al que llegó. Esta ruta debe quedar destacada hasta que se haga *click* en otro botón con el texto "`Finalizar Predicción`".

Para agregar aún más dinamismo, mientras haya una ruta destacada por predicción en los árboles, al hacer *hover* sobre un nodo hoja se debe destacar solamente la ruta hasta ese nodo, y al salir de ese nodo debe volver a destacarse la ruta de la predicción anterior. Es decir, solo se puede olvidar la ruta destacada al hacer *click* sobre "`Finalizar Predicción`".

[3] Pueden revisar los atributos usados por los nodos intermedios para determinar cuales son esos parámetros

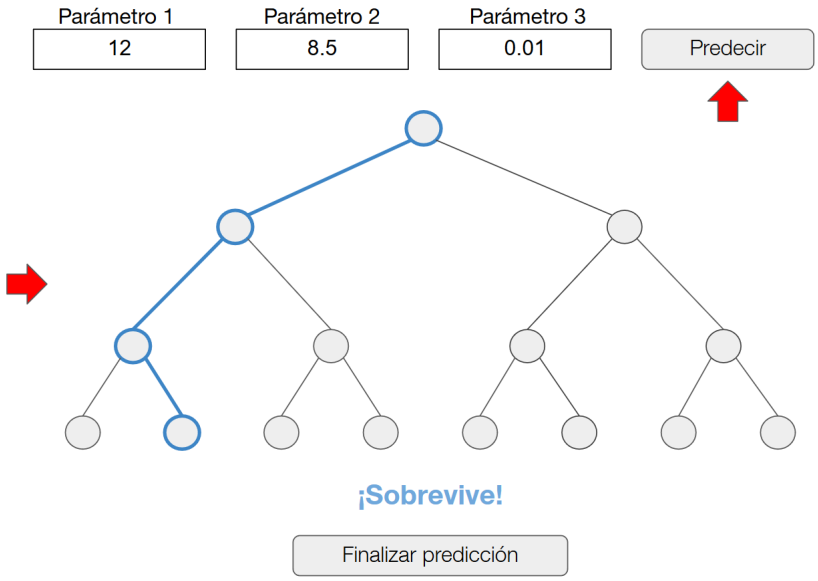


Figura 6: Ejemplo de posible *layout* del bonus

4. Presentación de resultados [1 punto]

Para esta parte, se permitirán dos, y sólo dos, formatos de entrega:

4.1. Formato Clásico: HTML + JS + CSS

4.1.1. Informe

Debe crear un informe donde presente los resultados obtenidos en esta tarea. Este debe ser un documento **HTML** que contenga lo siguiente

1. Todas las visualizaciones solicitadas anteriormente
2. Una breve explicación de las decisiones de diseño (visual encoding[4]) tomadas a lo largo del desarrollo de su tarea

Recuerde hacer uso de los diferentes *tags* que posee **HTML** para mostrar la información de forma ordenada y amigable para el lector. Se espera que al menos el documento contenga lo siguiente:

- Contenido centrado (textos e imágenes) y texto justificado.
- Cambios adicionales a un archivo **CSS** correspondiente, que enriquezcan la presentación de su informe.

No olvide que este documento será su informe de la tarea, por lo que adicionalmente se evaluará que cumpla con una correcta redacción y ortografía.

[4] *How según framework de Tamara Munzner*

IMPORTANTE: No se permitirá correr un localhost para abrir los datasets (`python -m http.server 8000`), en su lugar deben cargar los datos desde los links entregados directamente con D3. Pueden revisar la **ayudantía 6** para ver como lograr ésto.

Además, puede incluir un archivo `README.md`, pero solo se evaluará como informe lo que está contenido en su **HTML**.

4.1.2. Formato

En esta tarea debe respetar los siguientes formatos:

- Entregar *scripts* escritos en **JavaScript** en archivos separados al código **HTML**.
- Debe utilizar la versión 5 de **D3.js** y programar con un paradigma **declarativo**. Por ejemplo, no se espera ver un `for` para hacer agregar cada elemento **SVG**, sino que utilizar el *data-join* visto en ayudantías para agregar elementos.

4.1.3. Entregables

Usted debe entregar su tarea en el repositorio que se le crea automáticamente al ingresar al **enlace** señalado anteriormente. Los entregables que se esperan para éste formato de tarea son:

- **Presentación resultados**: un informe en formato **HTML** que detalle lo realizado y logrado en las anteriores secciones y un archivo **CSS** que enriquezca su presentación.
- `scripts.js`: *scripts* (Pueden ser varios) **JavaScript** que haga uso de **D3.js** para generar las visualizaciones solicitadas (Árboles y Stacked Bar Charts).
- `README.md`: un archivo en formato **Markdown** en caso de detallar alguna instrucción previa para visualizar la tarea, como la necesidad de ejecutar un archivo Python para preprocesar los datos. Esto es opcional.

4.2. Formato Pro: ObservableHQ

4.2.1. Informe

Debe crear un informe donde presente los resultados obtenidos en esta tarea en ObservableHQ. Este, al igual que en el otro formato, debe ser un documento que contenga lo siguiente

1. Todas las visualizaciones solicitadas anteriormente.
2. Una breve explicación de las decisiones de diseño (visual encoding[5]) tomadas a lo largo del desarrollo de su tarea. Esta parte debe ser escrita con código **HTML** y no código **Markdown**.

Se espera, además, que al menos el documento contenga lo siguiente:

- Contenido centrado (textos e imágenes) y texto justificado.

[5] *How según framework de Tamara Munzner*

No olvide que este documento será su informe de la tarea, por lo que adicionalmente se evaluará que cumpla con una correcta redacción y ortografía.

4.2.2. Formato

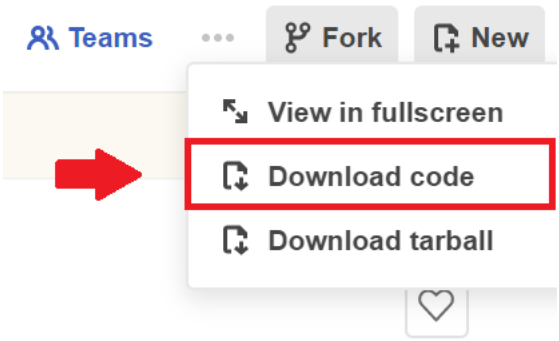
En esta tarea debe respetar los siguientes formatos:

- Debe utilizar la versión 5 de **D3.js** y programar con un paradigma **declarativo**. Por ejemplo, no se espera ver un `for` para hacer agregar cada elemento **SVG**, sino que utilizar el *data-join* visto en ayudantías para agregar elementos.

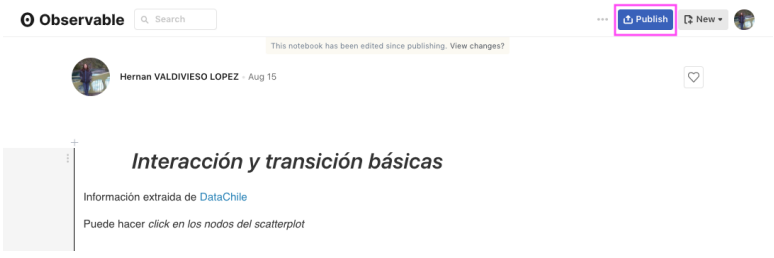
4.2.3. Entregables

Usted debe entregar su tarea en el repositorio que se le crea automáticamente al ingresar al **enlace** señalado anteriormente. Los entregables que se esperan para éste formato de tarea son:

- Código de su documento realizado en Observable, el cual pueden descargar en esta parte del documento:



- `README.md`: un archivo en formato **Markdown** con el *link* a su documento Observable. No olvides oprimir en *Publish* después de hacer un cambio para tener la versión más actualizada.



Importante: Cuando los ayudantes corrijan la tarea, ellos volverán a bajar el código con el botón *Download code* y verán las diferencias entre dicho código y el entregado por ustedes en el repositorio. Si se detecta alguna diferencia entre ambos códigos, **su tarea será evaluada con nota mínima y no podrá apelar**. Por lo tanto, luego de entregar la tarea, **no oprima en *Publish***.

5. Política de atraso

En la eventualidad de entregar pasada la fecha de entrega, se aplicará un **descuento** a la nota final obtenida en su tarea.

De haber atraso, el descuento comienza desde las 5 décimas. El descuento aumenta linealmente hasta las 24 horas posteriores del plazo inicial hasta un total de 20 décimas. Pasado un día del plazo inicial, el descuento es de 70 décimas, es decir, nota final **1**. La figura 3 muestra la función de descuento en función a las horas de atraso.

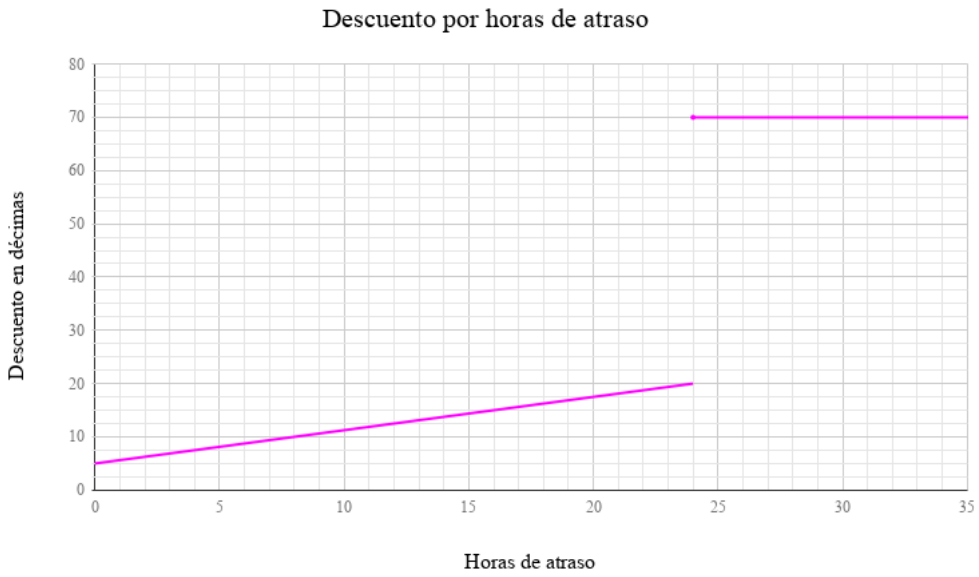


Figura 7: Descuento en nota según horas de atraso.