Web Intelligence & Big Data

Nicolás Hock Isaza - 200727001010

El curso se basó en cómo usando muchos datos (big data) se pueden encontrar y utilizar reglas para mejorar las experiencias de los usuarios en la web.

El ejemplo clásico es un buscador. Todo el mundo piensa en Google cuándo se habla de Big Data y de buscar en la web. Pero esta no es la única aplicación que tiene. Por ejemplo, las recomendaciones en Amazon son producto del mismo análisis y procesamiento de millones de pedidos y peticiones, que se procesan teniendo en cuenta la historia (reciente) del usuario en la página y se muestran productos que el usuario debería (querer) comprar.

El curso se dió en un formato de:

- Observar (Search)
- Escuchar (Machine Learning)
- Aprender (Information Extraction)
- Conectar (Reasoning)
- Predecir (Data Mining)
- Corregir (Optimization)

Todo este proceso apoyado por tecnología de "big data" (para manejar muchos datos) que permite obtener cierta información de los datos y poderlos manejar en muchos servidores a la vez.

En cada sección se introducían nuevos conceptos y se iban conectando para al final tener una "cadena" de herramientas que posibilitan tener una aplicación web "inteligente".

Observar y Buscar

En esta sección del curso se trataron diferentes formas de crear índices. El índice es igual al de un diccionario. Dada una palabra que estoy buscando (en el diccionario), me dice qué páginas web tenían esa palabra. El reto es poder crear un índice **muy** grande, donde la búsqueda sea rápida y hacerlo se demore un tiempo "aceptable".

Se sabe que buscar **una** palabra en un índice de *m* palabras es `O(log m)`, pero si se tiene una búsqueda de varias palabras (*q* palabras diferentes) es necesario armar o reunir el resultado. Si se tienen *r* resultados parciales (al menos una de las *q* palabras aparecen en un documento, que en total suman *r* documentos), para construir el resultado final se necesita `O(rq)`, pues hay que ir por cada palabra en cada resultado y preguntar si está.

Una forma básica de crear un índice es la siguiente:

```
class Index
  def create(D):  // D = conjunto de documentos
    for d in D:
      for w in d:
        i = index.lookup(w)
          if i < 0:
            j = index.add(w)
            index.append(j, d.id)
          else:
            index.append(i, d.id)
```

Si se tienen `n` documentos, `m` palabras en el alfabeto y `w` palabras por documento en promedio:

- Cada palabra de cada documento se debe leer una vez, el orden es **al menos** `O(nw)`

Adicionalmente, mientras *cada* palabra se lee:

- Necesitamos mirar si una lista asociada a esta palabra ya existe en el índice `O(log m)`
- Se inserta la URL (id) del documento en la lista (que se crea de ser necesario). Se puede asumir que esto tiene un costo constante.

Así, la complejidad para crear este índice es: `O(nw log m)`

Similitud vs Importancia

Cuándo se tiene una búsqueda (sobre todo de varias palabras), es posible obtener muchos resultados (suponiendo que hay entre 30 y 40 billones de páginas en Internet - Indexadas por Google). Entonces los resultados se deben presentar por similitud a la búsqueda (número de palabras que había en el documento) o por importancia de una página web (como el NYT o CNN)?

Para esto se utilizan diversos algoritmos, uno de ellos es el **Google Page Rank** que mide la importancia de una página en internet.

Page Rank se basa en la forma como funciona la web. Muchísimos documentos enlazados por vínculos, por el cual se pasa de un lugar a otro. Google Page Rank

toma un usuario de internet aleatorio, que visita una página aleatoria y sigue enlaces aleatorios en esa página. Entonces, cuál es la probabilidad que este usuario aleatorio visite una página?

Es claro que si una página tiene **muchos** enlaces, va a tener una probabilidad mayor, lo que la hace más importante y tiene un Page Rank grande. Pero, es importante aclarar que no solo los enlaces que entran a una página son suficientes para calcular el Page Rank, pues imaginemos que haya un ciclo en la web (que hay muchos!) y el usuario se queda allí para siempre.

Google maneja esto de diferentes maneras, una de ellas es hacer un "salto" con el usuario, donde no visita un enlace de la página sino otra página aleatoria (y vuelve a empezar la navegación aleatoria).

Un problema muy grande (e interesante) que hay con Google, es que mientras más efectivo sea (la gente *siente* que encuentra lo que quería, sin esfuerzo), va perdiendo precisión. La razón es que las personas empezarán a **no** usar enlaces a páginas pues "es más fácil buscar en Google", lo que hace que las páginas nuevas (y el contenido nuevo) no sea vinculado y el Page Rank pierde precisión!

Es también importante resaltar que las personas somos muy malas para recordar hechos específicos y cada vez dependemos más y más de Google, entonces al **buscar y encontrar** (recuperar la información), le estamos dando información a Google sobre la relevancia de un documento en nuestra búsqueda. Esto es darle retroalimentación a Google, que mejora el Page Rank.

Otros tipos de búsquedas

La búsqueda de palabras en texos es, posiblemente la más sencilla. La dificultad se obtiene por la cantidad de información, no por la forma de buscar. Pero, qué pasa si lo que buscamos no son palabras sino objetos? Por ejemplo, cómo se buscaría una imagen en una colección de imágenes? Un índice **no** es la mejor forma de hacer esto. Existe una aproximación llamada **Locality Sensitive Hashing**.

Locality Sensitive Hashing

La idea básica de LSH es que los objetos que "se parecen" tendrán el mismo resultado en una función de Hash `h`. Es decir:

- Si `x == y` o `x similar y`, `h(x) == h(y)` con gran probabilidad.
- Si `x != y` o `x muy-diferente y`, `h(x) != h(y)` con gran probabilidad.

La construcción de la función `h` es compleja y estaba por fuera del alcance del curso, pero se recomienda leer Ulman and Rajamaran - Capítulo 3.

Normalmente, LSH se verá de la siguiente forma:

$$1 - (1 - p^k)^b$$

Donde `k` es las veces que se repite cada función, `p` la probabilidad de que la función evaluada sea 1 (un acierto) y `b` la cantidad de funciones que se toman. Esto implica que una variación pequeña en `p`, afectará de manera muy grande el resultado.

Escuchar

En esta sección se trabajó el tema de *información*. Se tocaron varios aspectos de teoría de la información, como "¿qué es nueva información?" y se brinda la definición de Claude Shannon en 1948: La información está relacionada a la *sorpresa*. Es decir, un mensaje que informa de algo que tiene una baja probabilidad `p` tiene más información que algo que **siempre** ocurre.

Por *escuchar* se entiende, por ejemplo, el poder diferenciar usuarios que van a visitar una página web a usuarios que van a comprar. Se "escucha" sus búsquedas y se toman decisiones respecto a estas. Ojo, es diferente a "encontrar" información. Si hay una gran probabilidad de comprar (se buscó por ejemplo 'flores rojas baratas') se puede mostrar publicidad. Pero, si la búsqueda es 'flores rojas', es posible que **no** se esté pensando en comprar.

Al tomar en cuenta esto, se introduce el concepto de "información mutua" (se definirá más tarde), al aumentar la información mutua entre un usuario (su búsqueda) y la publicidad, se aumenta la rentabilidad.

Del mismo modo, el contenido de las páginas web sirve como "búsqueda" para mostrar la publicidad. Pero entran a jugar otros factores como las palabras claves de un documento y se introduce el concepto de **Inverse Document Frequency - IDF** de

una palabra. *Es inverso pues no se buscan documentos desde palabras claves (que sería la búsqueda), sino palabras claves a partir de un documento*. Esto es, si una palabra es muy común en la web, no es importante. Así se tiene que la palabra "*Turing*" es mejor palabra clave que "*Persona*". Pero, palabras más raras, más comunes en el documento, son aún mejores. Entonces se define la **frecuencia de un término - TF** y se puede obtener el **TF-IDF**.

IDF de una palabra `w` = $IDF = \log_2(N/N_w)$

Frecuencia de `w` en un documento `d` = $TF = n_w^d$

El **TF-IDF** = $n_w^d * \log_2(N/N_w)$

El **TF-IDF** se inventó como una heurística, pero en 2003 se demostró que la **información mutua** entre **todas** las páginas y **todas** las palabras es proporcional a:

$$MI = \sum_d \sum_w n_w^d * \log_2(\frac{N}{N_w})$$

- "*An information-theoric perspective of TF-IDF measures*", Kiko Aizawa, Journal of Information Processing and Management, Volume 39 (1), 2003

Machine Learning

Machine learning es "aprender" basado en (mucha) información pasada. Es hacer predicciones usando probabilidad condicionada (por ejemplo, el teorema de Bayes).

Es muy importante resaltar que si dos variables `R` y `B` son **independientes**, se tiene que `P(R | B) = P(R)`. Es decir, el hecho de *saber* que `B` ocurrió, no influye en la probabilidad de que ocurra `R`.

Clasificador Bayesiano Ingenuo

Emepecemos por definir un clasificador, esto es simplemente un mecanismo de clasificar algún contenido. Por ejemplo, se puede asignar un "sentimiento" positivo o negativo a un comentario (Me encanta la leche - positivo vs No me gusta la leche - negativo).

Un Clasificador Bayesiano es aquel que se basa en el Teorema de Bayes. Es *ingenuo* pues se **asume** que todas las variables predictorias son independientes. Recordemos el Teorema de Bayes:

`P(B,R) = P(B|R) P(R) = P(R|B) P(B)`

Ahora, supongamos que un usuario busca "flores `R` ojas y `B` aratas", queremos saber si ese usuario es un 'navegador' o un 'comprador'. En otras palabras si `C = y`.

- `P(C|R,B) * P(R,B)` -> Cuál es la prob. de que sea un `C` omprador si se da `R` y `B`?
- `P(R,B|C) * P(C)` -> Bayes
- `P(R|B,C) * P(B|C) * P(C)` -> Bayes en el primer término
- `P(R|C) * P(B|C) * P(C)` -> Independencia de términos

Ahora, si se dan valores a `R` y `B` (`r` y `b`), se tiene:

$$\frac{p(r|C=y) * p(b|C=y) * p(C=y)}{p(r|C=n) * p(b|C=n) * p(C=n)}$$

Recordando que *machine learning* es basando en datos antiguos, se tienen los valores de cada término de la expresión.

Estos clasificadores funcionan igual para N variables (debido a la independencia entre ellas), entonces se puede generalizar a:

$$L = \prod_{i=1}^{N} \frac{p(x_i|C=y)}{p(x_i|C=n)} * \frac{p(C=y)}{p(C=n)}$$

Como esto da una probabilidad, se elige un límite. Si la probabilidad es al menos el límite se tiene que el usuario es un comprador, si da por debajo se tiene que el usuario es un navegador.

Si miramos *machine learning* desde el punto de vista de teoría de la información, se están transmitiendo características (de una búsqueda, como flores rojas baratas) para predecir un comportamiento (comprar o solo navegar). La idea es mejorar la información mutua entre la búsqueda y el resultado, pero para esto, se necesita definir

información mutua. Entonces, la **información mutua** entre un término (`F`) y una comportamiento (`B`) es:

$$I(F,B) = \sum_{f,b} p(f,b) \log \frac{p(f,b)}{p(f) * p(b)}$$

Claramente, si los términos son independientes, se tiene que `p(f,b) = p(f)*p(b)` y `I(F,B) = 0` . Se presenta un ejemplo en el cual se tienen comentarios sobre el curso, varios positivos y negativos (ya catalogados, recordemos que es decidir en base a resultados pasados). Se muestra como la palabra "odiar" tiene información mutua con el sentimiento (negativo), mientras que la palabra "curso" no.

Cargar Datos

En la sección de cargar de datos, se habla de computación paralela. Se empieza definiendo la **aceleración (S)** que nos dice realmente cuánto se mejoró al paralelizar el procesamiento. Si procesar los datos con 1 solo proceso toma `Ti` tiempo y con `p` procesos toma `Tp` , entonces `S = Ti / Tp` .

Se define también la **eficiencia (E)** de la paralelización como: `E = Ti/(p*Tp)` .

Map Reduce

Map Reduce es una *forma* de paralelizar el procesamiento que consiste en dos *entidades*. Un *mapper* y un *reducer*. Los *mappers* reciben parejas `K1, V1` de un diccionario (llave -> valor) y lo transforman en parejas `K2, V2` . Los *reducers* reciben **todas** las parajas (`K2, V2`)de un `K2` específico y procesan estos datos. Normalmente se combinan los datos (contar ocurrencias o sumar números). La plataforma donde se corre el sistema está encargada de enviar todas las parejas de `K2` al mismo *reducer*.

Es importante resaltar que *map-reduce* es una forma de procesamiento **en lote**. Esto quiere decir que no hay interacción (ni feedback) hasta que se tiene el resultado final.

Tomemos el ejemplo de contar las palabras en docuementos. Queremos saber cuántas veces existe una palabra en un conjunto de documentos. Tenemos entonces

que cada *mapper* recibe una pareja de `(idDocumento, 'texto')` y producen un arreglo de parejas `(palabra, cuentaPalabraEnDocumento)`.

$$(d_k,' w_1 ... w'_n) n \rightarrow [(w_i, c_i)]$$

Ahora, cada *reducer* redice un arreglo de parejas, donde el primer elemento de cada pareja es el mismo (la palabra). Claramente esto viene de muchos documentos diferentes, pero es **necesario** que la *llave* de la pareja sea la misma. Tenemos entonces el siguiente reducer:

$$(w_i, [c_i]) \rightarrow (w_i, \sum_i c_i)$$

El proceso se puede mejorar un poco más si los *mappers* procesan ciertos datos y pasan resultados parciales a los reducers. Pero, para que esto sea posible, la función final debe ser asociativa y conmutativa. Esto se debe a que se hace todo en diferente orden y por diferentes *mappers*, entonces el orden no puede importar.

La solución al exámen de Map-Reduce se puede encontrar en: https://gist.github.com/8d43d211816208efe7d2. El archivo `word_count.py` contiene la solución al problema.

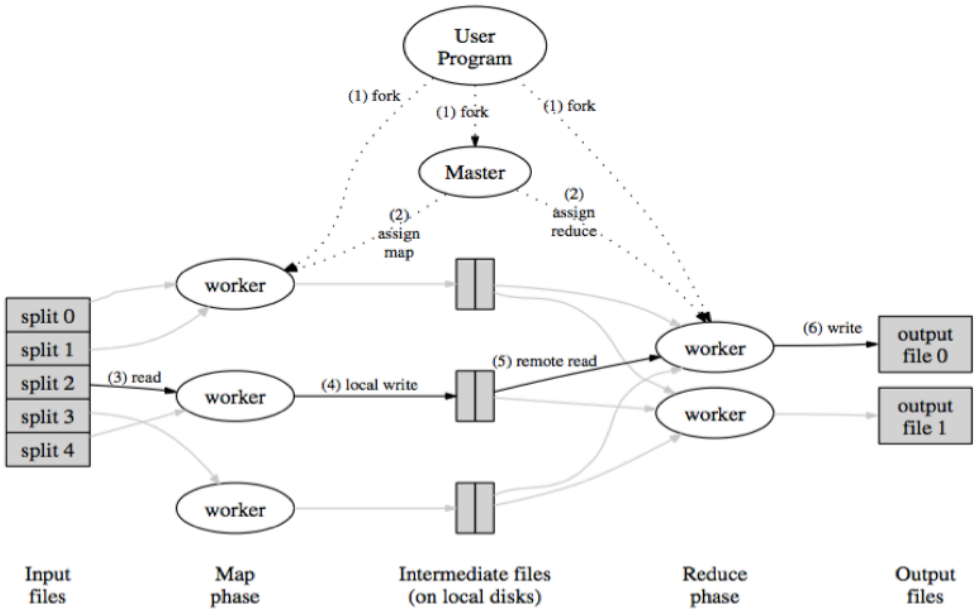
¿Cómo funciona Map Reduce?

Se tiene un nodo principal (master) que vigila el proceso. Se tienen los archivos de entrada divididos en la red (para evitar un cuello de botella en la lectura). El proceso master inicia los *mappers*, que leen los archivos y los procesan. Es importante resaltar que, por la forma como se plantean los problemas, si un mapper falla se puede iniciar otro y volver a empezar sin que el sistema tenga problemas.

El problema está si falla el nodo master. Obviamente este nodo está monitoreado y en una máquina de alto rendimiento y, dados `n` (grande) procesos, la probabilidad de que al menos uno falle es muy alta, pero de que falle uno específico es muy baja. Basados en esto el master "siempre estará vivo".

Los *mappers* escriben los resultados (parciales) a disco y el master inicia varios nodos *reducers*. Un *mapper* que haya finalizado se puede utilizar como un *reducer* posteriormente. El master es el encargado de enrutar los resultados parciales para que

cada *reducer* reciba **todas** las parejas de la misma llave y así poder juntar los resultados.



Map Reduce es muy una aproximación muy poderosa en la cual **siempre** se escribe (o se obtiene) nueva información. Es importante entender que el resultado de un proceso de Map-Reduce puede ser la entrada de otro proceso, encadenando así diferentes resultados para generar mucha más información **nueva**.

Aprender

Cuando se habla de aprender, se hace referencia a extraer "reglas" desde los datos. Estas reglas se dan por relaciones entre diferentes características de los datos, que no tienen que ser independientes entre ellas. Entonces, aprender esas reglas es básicamente encontrar las características de los datos que están relacionadas.

Por ejemplo, se podría "aprender" que un comentario que tenga "gusta mucho" es un comentario positivo, mientras que uno que tenga "no gusta" es negativo. Se han conseguido otras reglas menos obvias como que, si un cliente compra leche y pañales, probablemente comprará cerveza.

El razonamiento para obtener una relación entre dos términos (características) es algo así: Si los dos términos (por decir pañales y cerveza) fueran independientes, entonces no habría muchos más datos donde no ocurrieran juntos que donde sí. Esta cantidad de ocurrencias donde salen los dos muestra una fuerte relación entre ellos. Esto es una razón necesaria, pero no suficiente para obtener una regla.

Para inferir la regla asociativa `A, B, C => D` se necesita:

- **Alto soporte:** `P(A,B,C,D) > s`, esto quiere decir que los términos aparecen juntos "*suficientes*" veces.
- **Alta confianza** `P(D|A,B,C) > c`, esto quiere decir que de todos los casos donde se tiene `A,B,C`, muchos tienen `D`.
- **Altamente Interesante** `(P(D|A,B,C) / P(D)) > i`, esto significa que la *confianza* de que `D` ocurre más con `A,B,C` es mayor que la probabilidad de obtener `D` solo. Si por ejemplo, todos los datos tienen `D`, la regla no es *interesante* y no se debe tener en cuenta.

Ahora miremos **cómo** se encuentran estas reglas, sabiendo que existen muchas combinaciones de términos, si cada una tiene dos valores (por ejemplo, se compró o no), se tienen `2^N` combinaciones, lo cual es muy grande.

Se necesita una observación clave: **Si `A,B` tienen alto soporte (`> s`), entonces también lo tienen `A` y `B` (por separado)**. Esto permite crear un algoritmo que busque todos los términos (separados) que tengan soporte `> s` y este set va a contener **todas** las parejas, tripletas y demás conjuntos posibles. Así, no se necesitan mirar **todas** las combinaciones de términos, sino las *posibles*.

Es claro que al obtener todas las parejas `X,Y` que tienen soporte `> s`, se obtienen todas las posibles tripletas en ese mismo set. Y así sucesivamente hasta no encontrar un conjunto de `N+1` términos con soporte `> s`.

Al tener todos los posibles valores (parejas, tripletas… - que no van a ser muchos) se mira, para cada set, la *confianza* y que tan *interesantes* son.

Como solo se necesita contar, se puede utilizar Map Reduce

¿Cuánto se ha aprendido?

Se plantea una pregunta interesante y que es un tema abierto de investigación y es,

realmente, ¿cuánto se ha aprendido?. Si tomamos por ejemplo un traductor que toma un texto en inglés y lo traduce a chino, siguiendo unas reglas de gramática y semántica, se puede decir que el traductor sabe chino?

Todo esta traducción se da por un "*razonamiento mecánico*" y se cuestiona hasta que punto esto puede ser considerado "inteligencia y aprendizaje".

Conectar

Este *conectar* es poder pasar más allá de aprender, es razonar y poder decir "*por qué*" de algo. Utilizando un poco la lógica y realizando razonamientos tanto deductivos (de general a individual) como inductivos (de individual a general).

Si tenemos, por ejemplo la pregunta "*¿Quién es el líder de E.E.U.U?*" y tenemos muchos hechos (*facts*) como [X es primer ministro de Y], [X es presidente de C] y por algún método podemos aprender que `X es presidente Y => X líder Y`, podríamos saber que el líder de E.E.U.U es Obama al tener que `Obama es presidente de E.E.U.U`. Pero, esto trae muchos problemas, por ejemplo en India hay primer ministro y presidente. ¿Cuál de los dos es el líder? Se necesita *más conocimiento* para responder esto.

El problema con la lógica *convencional* es que para poder **asegurar** algo de este estilo, se necesitan cuántificadores universales (no existenciales - `Para todo x` y no `Existe un x`). Esto es muy poco común en la vida real. Por ejemplo, si tenemos:

- Para *casi todos* los x, `A(x) => B(x)` - "*La mayoría de los bomberos son hombres*"
- Para *casi todos* los y, `B(x) => C(x)` - "*La mayoría de los hombres tienen trabajos de bajo riesgo*"

No se puede concluir que para *casi todos* los x, `A(x) => C(x)` - "*La mayoría de los bomberos tienen trabajos de bajo riesgo*".

Aún cuándo las primeras dos afirmaciones son ciertas, no se puede deducir la tercera (no existe esa transitividad). Gráficamente se tiene algo así:

Otro problema con la lógica se da en casos como este:

- Si el aspersor (con lo que se riega un jardín) estaba prendido, entonces la manga está mojada (`S => W`)
- Si la manga está mojada, entonces llovió (`W => R`)

Obviamente no se sigue que `S => R`, el problema aquí es que la *causalidad* se manejó de manera diferente. Es decir, el hecho de que la manga esté mojada (`W`)es una **característica** observable de `S` y de `W`. Afirmar que *Si* `W` *pasó es porque pasó* `R` es una *abducción*.

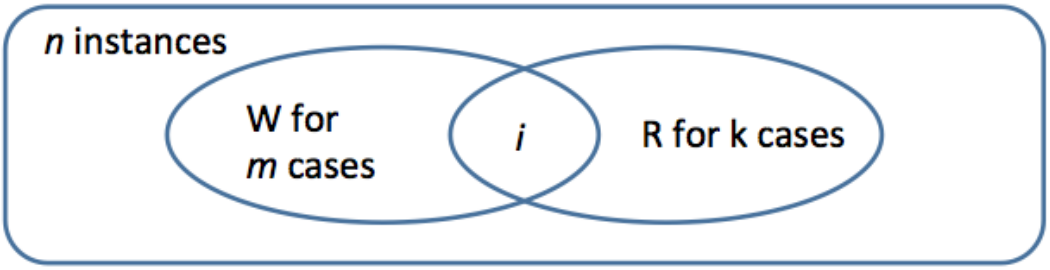
Aquí se introduce el concepto de razonamiento abductivo que busca encontrar la mejor explicación o la más probable. En el ejemplo podría ser la *lluvia* (dado que en Colombia llueve mucho).

Tablas de probabilidad y marginación

Si tenemos una tabla de valores:

ID	W	R
1	y	n
2	y	y
3	n	n
…	…	...

Considerando `n` casos totales, donde `W` pasa en `m` casos, `R` pasa en `k` casos y en `i` casos pasan tanto `W` como `R`.



Se puede crear una tabla de probabilidades para `p(R,W)` de la siguiente manera:

R	W	P
y	y	`i/n`
n	y	`(m-i)/n`
y	n	`(k-i)/n`
n	n	`(n-m-k+i)/n`
		$p(R,W) = T^{R,W}$

Pero supongamos que queremos encontrar la probabilidad de que llovió. Uno puede recorrer la tabla grande, contar las ocurrencias donde llovió y dividir por el total, o se puede utilizar la tabla más pequeña donde ya se tienen algunos cálculos. Pero como la tabla pequeña tiene en cuenta `W`, hay que *marginarla*, esto es, sumar las entradas donde los valores de `R` sean iguales (sin importar los valores de `W`). Así, se tiene que:

R	P
y	`k/n`
n	`(n-k) / n`

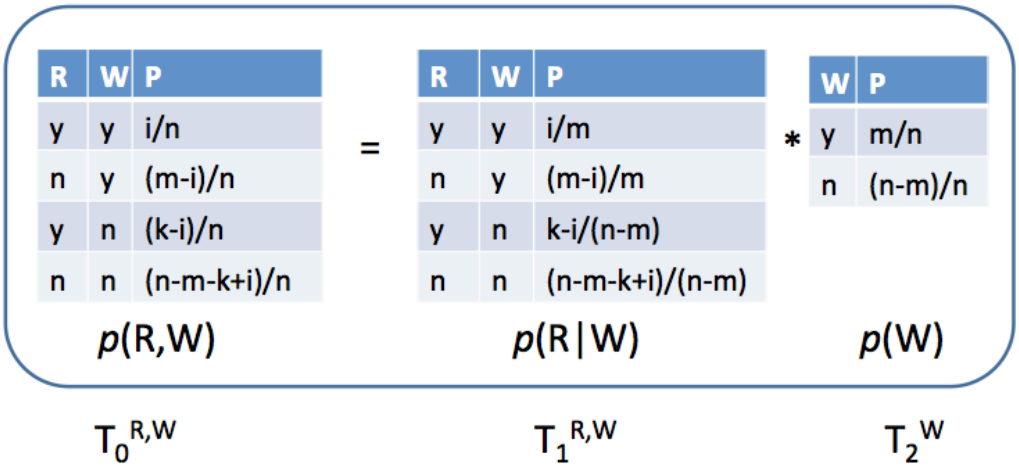
Ahora, miremos que en SQL, la marginación es equivalente a sumar una columna:

$$\sum_{w} p(R,W) =_R G_{SUM(P)} T^{R,W}$$

```
SELECT R, SUM(P) from T GROUP BY R
```

Tablas de probabilidad y Teorema de Bayes

Ahora, con el teorema de Bayes nos dice que `p(R,W) = p(R|W) * P(W)`, si tenemos las tablas pequeñas podemos generar la tabla de *probabilidad conjunta* (`p(R,W)`) con un `JOIN` en los atributos comunes (en este caso, `W`). Gráficamente es más fácil de apreciar:




```
SELECT R, SUM(P1*P2) from T1, T2 WHERE W1=W2 GROUP BY R
```

Recordemos que la idea es siempre poder construir la información de tablas más grandes a partir de tablas más pequeñas.

Tablas de probabilidad con *evidencia*

La **evidencia** son características observadas. Por ejemplo, podemos afirmar que la manga está mojada (`W=y`). Esto va a alterar las probabilidades con las que se trabajan, pues ahora buscaremos `P(R|W)`.

Supongamos que tenemos una probabilidad conjunta (`p(R,W)`) y podemos observar que la manga está mojada. Necesitamos restringir la tabla actual para que tenga solo las entradas que nos sirven (que cumplen con la observación o la

evidencia). A esto se le conoce como la *aplicación de la evidencia*.

Usando nuevamente Bayes, tenemos que:

$$p(R,W)e^{W=y} = p(R|W=y) * p(W=y)$$

En SQL aplicar la evidencia es utilizar *select*: `SELECT R,W,P from T WHERE W=y`

Entonces tenemos que la probabilidad a posteriori (probabilidad condicional que es asignada después de que la evidencia es tomada en cuenta), de que haya llovido (`R`) dada la evidencia (`e`) es:

$$p(R|e^{W=y}) = \frac{p(R,W)e^{W=y}}{p(e^{W=y})}$$

En la tabla sería:

A	P
y	`i/m`
n	`(m-i)/m`

Lo más importante es resaltar que construir probabilidades conjuntas en SQL es realizar JOINS y aplicar evidencias es hacer SELECT

Redes Bayesianas

Miremos como utilizar el clasificador bayesiano básico con tablas de probabilidad. Si tenemos estas tablas:

p(W|R)

W	R	P
y	y	0.9
n	y	0.1
y	n	0.2
n	n	0.8

p(R)

R	P
y	0.2
n	0.8

Se puede saber la probabilidad de que haya llovido si vemos que la manga está mojada así:

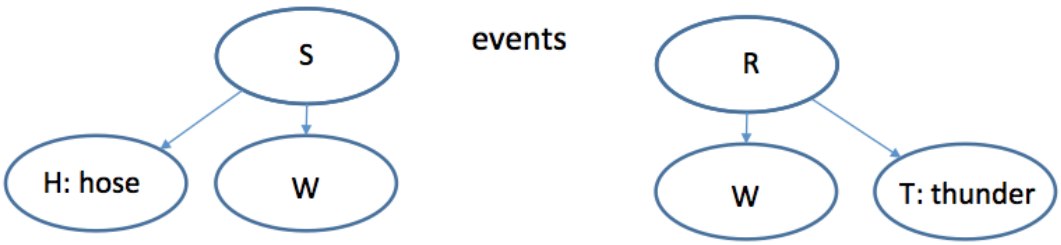
```
SELECT R, SUM(P1 * P2) FROM T1, T2 WHERE W=y AND R1 = R2 GROUP BY R
```

Esto nos da la tabla

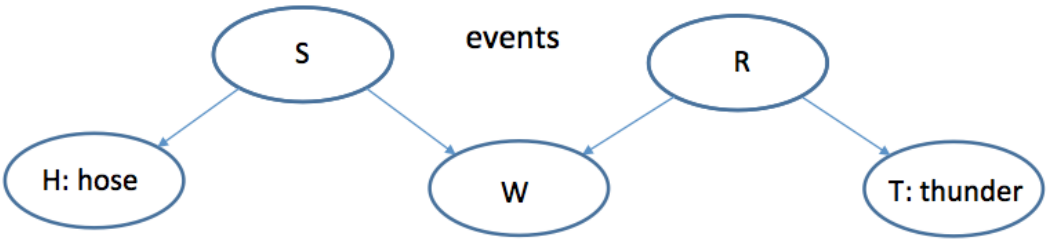
R	P
y	0.9 * 0.2 = 0.18
n	0.2 * 0.8 = 0.16

Normalizando tenemos que `p(R=y) = 0.18 / (0.18 + 0.16) = 53%`.

Ahora supongamos que tenemos más características observables, por ejemplo que haya truenos (`T`) o que se pueda ver una manguera (`H`) en el piso. Podemos tener dos clasificadores para esto:



Pero, como `W` **es la misma observación**, debemos juntar los dos clasificadores en uno solo. Esto se conoce como una red bayesiana.



Tenemos que, por Bayes (aplicada varias veces):

```
p(H,W,T,S,R) = p(H,W,T,S|R) * p(R) = p(H,W,T|S,R) * p(S|R) * p(R)
```

Como S y R son independientes (y se asume que las observaciones `H,T,W` son independientes dados `S,R`), entonces:

```
p(H,W,T,S,R) = p(H|S,R) * p(W|S,R) * p(T|S,R) * p(S) * p(R)
```

Como H es independiente de R y T es independiente de S, se tiene que:

```
p(H,W,T,S,R) = p(H|S) * p(W|S,R) * p(T|R) * p(S) * p(R)
```

Como regla general se tiene que *sólo* se necesita la probabilidad conjunta de los padres en el árbol

Esto quiere decir que se puede seguir utilizando SQL simplemente realizando los *JOIN*, *SUM* y *SELECT* necesarios para aplicar evidencias (*select*) o marginar alguna variable (*sum*) entre las diferentes tablas de probabilidad (*join*)

El hecho de que dos eventos puedan producir la misma observación tiene un efecto bastante interesante al analizar los datos. En nuestro ejemplo tenemos que tanto `S` como `R` causan `W` , podemos ver como la probabilidad de un evento dado que se observó la característica (en este caso `W`) disminuye aún sin observar el otro evento.

Es decir, el hecho de que haya otra manera de mojar la manga además de la lluvia (con el aspersor) disminuye la probabilidad de lluvia con solo ver la manga mojada, aún sin tener información de haber usado o no el aspersor.

Si se observa que un evento pasó, por ejemplo que el aspersor estaba encendido (`S=y`), la probabilidad de que haya llovido por que vemos la manga mojada baja aún más. Esto se llama **propagación de la credibilidad**.

Esta propagación de la credibilidad es la base para el razonamiento abductivo, que busca dar la mejor explicación posible para algo. Esto es una forma de trabajar con cierta incertidumbre.

No sobra mencionar que SQL, posiblemente, no es la manera más eficiente de trabajar con estas probabilidades. Map Reduce también puede ser utilizado.

Comentarios del curso

De los tres cursos de Coursera que realicé, este fue el mejor. Aún cuándo el tema es muy amplio y tiene infinidades de aplicaciones, los ejemplos utilizados eran muy interesantes. Las tareas eran prácticas y era un tema completamente nuevo para mí.

En cuánto a los quizzes y exámenes, había ciertos problemas con algunas calificaciones. Por ejemplo, había preguntas con múltiples respuestas válidas. Preguntaban por una probabilidad que daba, por ejemplo, 2.3 y había respuestas que eran "Entre 2 y 3" y "Menor que 3". Ambas son válidas. Pero, a pesar de esto, las tareas eran completamente relevantes con el contenido de los vídeos.

Los ejemplos y la forma como se llevaba el tema era muy bien planeado y las explicaciones eran muy claras. Eso sí, sigue siendo un abre-bocas para el tema, pero es algo que en 8 semanas es muy difícil de cubrir. Más en una clase tan "impersonal".

Otro problema (que al final se resolvió) era que las diapositivas no estaban disponibles pues son contenido de un libro que el profesor está escribiendo y el editor no le permitió compartirlas hasta el final. Aún así, como los vídeos y las preguntas de los exámenes estaban altamente ligados no había mucho problema con eso.

Algo que era muy bueno, es que en los diferentes temas del curso, el profesor iba diciendo ciertos trabajos de investigación que se están haciendo actualmente.