Guion de prácticas

Shopping5

Recomendando Marcas : Matrices Bidimensionales

Mayo de 2021

UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM-GII-GADE

Curso 2020/2021

Índice

1. Descripción

Las prácticas anteriores están orientadas al análisis del registro de actividad de los clientes de una web de venta de productos y la elaboración de informes de comportamiento de las ventas. En esta práctica final ilustraremos cómo es posible utilizar los mismos datos para garantizar una mejor experiencia del cliente y que de camino nos garantice un incremento en las ganancias de la tienda (en las ventas).

La idea esencial es permitir que el sistema recomiende al cliente nuevos productos que les puedan ser de su interés. Por ejemplo, todos estamos acostumbrados a sistemas que nos recomiendan películas en plataformas digitales como NetFlix, HBO, etc. La idea es simple, mostrar al usuario un conjunto de películas que encajan con los gustos o preferencias de los usuarios, como se ilustra en la siguiente figura:

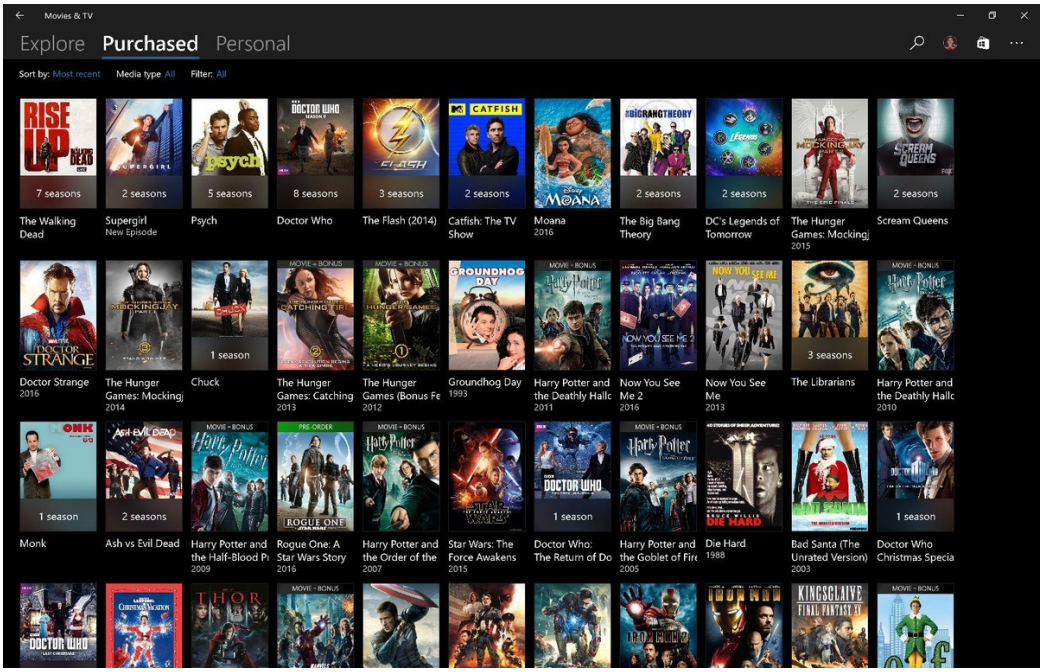


Figura 1: Sugerencias de películas

En este caso, y de forma transparente al usuario, nuestro objetivo es permitir a una tienda de comercio electrónico proporcionar (como lo gestione en la interfaz gráfica es otro problema) una recomendación de aquellos productos que les sean de más interés.

2. Arquitectura de la práctica (recordatorio)

En las prácticas anteriores ya se implementaron las clases `DateTime`, `Event`, `Pair`, `EventSet`, e `Index`, estas dos últimas implementadas utilizando memoria dinámica. Ha sido un trabajo progresivo que nos permitirá en parte abordar con mayor facilidad este último problema, el de recomendar a un cliente los mejores productos.

Continuaremos trabajando con estructuras dinámicas, en concreto nos planteamos el diseño e implementación de estructuras matriciales etiquetadas (se detalla más adelante en esta memoria) que permiten agrupar elementos en filas y columnas. Para ello deberemos de implementar una nueva clase, `Matrix`, que nos proporciona las herramientas para trabajar con este tipo de datos así como un programa principal que será el encargado de realizar las recomendaciones propiamente dichas.

Para abordar el problema de la recomendación nos planteamos utilizar uno de los algoritmos que más ha dado que hablar en los últimos años: El algoritmo PageRank.

2.1. Los datos

En esta práctica seguiremos trabajando con un pequeño fragmento del conjunto de datos reales (alrededor de 75,000 registros) descargados desde la plataforma Kaggle[1]. No detallaremos estos datos, pues asumimos que son más que conocidos[2].

3. Aplicando el PageRank a nuestra plataforma de comercio electrónico

El algoritmo PageRank[3] fue creado en 1998 por Sergey Bring y Larry Page, fundadores de la compañía Google, con la finalidad de determinar para cada página web su importancia dentro de la WWW y se considera, entre otros factores, el origen de dicho sistema de recuperación de información. Tras patentarlo Bring y Page cedieron a Google los derechos para utilizarlos. Una idea de la importancia del algoritmo es que solo 9 años más tarde las acciones se vendieron por más de 300 millones de dólares.

Pero, ¿Cómo se define la importancia de una página web?. Bring y Page consideraron que podrían estimarla al analizar la estructura de hiperenlaces entre páginas. Su hipótesis fue que cuando enlazamos una página web A con otra B estamos indicando que para A el contenido de B es importante (B recibe un `voto` positivo de A). Así, por un lado, la importancia de una página dependerá del número de páginas que la apuntan (votos que recibe). Pero por otro lado, también se considera que el valor de cada voto que recibe B dependerá también de la importancia de la página que le vota. De este modo, una página apuntada por varias páginas de calidad será más importante que otra apuntada por el mismo número de páginas, pero de menos calidad. Puede parecernos un galimatías, pero el algoritmo PageRank proporciona una solución a este problema.

Desde un punto de vista abstracto, el algoritmo PageRank recibe como entrada un grafo, donde los nodos (páginas web) estan interconectados

[1] Kaggle (Abrir en navegador →)

[2] Para más detalle consultar los guiones de las prácticas anteriores.

[3] Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. Computer networks and ISDN systems, 30(1), 107-117

mediante enlaces (hiperenlaces), y obtiene como salida una ordenación (`ranking` de nodos).

El algoritmo PageRank puede utilizarse en un amplio abanico de dominios, como por ejemplo en Twitter para mostrar a un usuario aquellos otros que podrían serle de interés (los nodos son usuarios y los enlaces la existencia de retweets); para ordenar espacios públicos al predecir posibles flujos de usuarios (nodos son las espacios (plazas, calles, ...) y los enlaces las interconexiones); predicción del rendimiento de atletas en competiciones (los nodos representan a los atletas que se unen mediante un enlace cuando uno gana a otro en una determinada competición), etc.

3.1. Definición de relaciones entre marcas

Con todo ello, no es de extrañar que el algoritmo PageRank también se haya utilizado en problemas de recomendación en sitios de comercio electrónico. Lo único que tenemos que definir es cómo se determina una relación entre dos ítems (productos, marcas, etc.). En nuestro caso particular estudiaremos las relaciones entre marcas[4].

Relación entre marcas: Una marca X está relacionada con una marca Y, a través del tipo **tipo** ($tipo \in \{\text{“}purchase\text{”}, \text{“}view\text{”}, \text{“}cart\text{”}, \text{“}remove\_from\_cart\text{”}\}$), y lo notamos como $R(X, Y, tipo)$, si existe al menos una sesión en el `evenSet` $\mathcal{E}$ en la que aparecen dos eventos distintos, $e1, e2$, de tipo **tipo** en la que aparecen X e Y, esto es, $R(X, Y, tipo)$ cuando existen $e1, e2 \in \mathcal{E}$ que satisfacen las siguientes tres condiciones:

- $e1.getSession() = e2.getSession()$
- $e1.getType() = tipo \wedge e2.getType() = tipo$
- $((e1.getBrand() = X \wedge e1.getBrand() = Y) \vee (e1.getBrand() = Y \wedge e1.getBrand() = X))$

Es importante notar que $R(X, Y, \text{“}tipo\text{”})$ es simétrica, y por tanto, si X está relacionado con Y, Y también lo estará con X.

Por ejemplo, consideremos los siguientes datos (nos centramos en los campos de interés) y asumimos que estamos interesados en el tipo `purchase`, esto es, buscamos las relaciones $R(X, Y, purchase)$: En este caso, tenemos por ejemplo que las marcas A y B están relacionadas pues los eventos de las líneas 6 y 9 cumplen todos las propiedades, mientras que A y C no están relacionadas ya que aun habiendo eventos en la misma sesión ($S1$) en las que están implicadas (líneas 6 y 11), estos no son del tipo `purchase`.

[4] Considerar los productos sería muy costoso (el número total de productos es unos 16000, por lo que el número posible de relaciones sería de $16000^2 = 256000000$ por lo que para almacenarlas todas (asumiendo un valor double necesitaríamos 16384000000 bits, esto es del orden de $1{,}907GB$ de memoria.

evs

	user,	type,	brand,	session
0	u1,	view,	A,	S1
1	u1,	cart,	A,	S1
2	u1,	view,	B,	S1
3	u1,	cart,	B,	S1
4	u1,	purchase,	A,	S1
5	u2,	view,	B,	S2
6	u2,	purchase,	B,	S2
7	u1,	purchase,	B,	S1
8	u2,	purchase,	C,	S2
9	u1,	view,	C,	S1
10	u3,	cart,	B,	S3
11	u3,	cart,	C,	S3
12	u3,	purchase,	B,	S3
13	u3,	purchase,	A,	S3
14	u2,	purchase,	D,	S2
15	u4,	purchase,	E,	S4
16	u1,	purchase,	B,	S5
17	u4,	purchase,	,	S4
18	u1,	purchase,	D,	S5

Figura 2: EventSet original

3.2. ¿Cómo obtenemos todas las relaciones entre marcas?

Para poder obtener todas las relaciones entre marcas en primer lugar debemos considerar los eventos ordenados por sesión. Este no es el orden original del eventSet, sin embargo podremos utilizar un índice como los que hemos implementado en la práctica anterior, donde la clave sea "Session".

Para permitir este índice solo tenemos que refactorizar el método `build` de `Index`, permitiendo un nuevo valor (le asociaremos el valor 2 a `_onBrand` para indicar que estamos construyendo un índice por sesión).

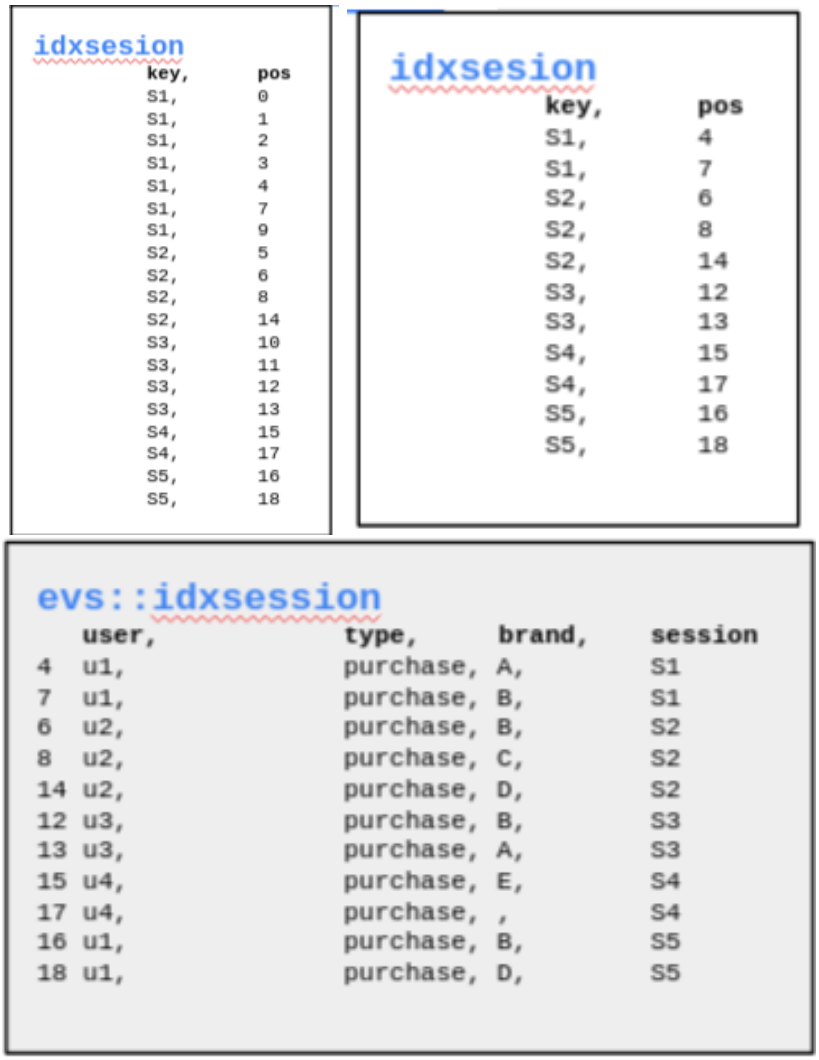
Una vez que tenemos el índice nos podemos quedar con el subíndice al filtrar los eventos del tipo que estamos interesados, en nuestro caso `purchase`. Para ello, también debemos refactorizar el método `rawFilterIndex` de la práctica anterior.

Un ejemplo de cómo puede quedar todo es:

```
EventSet evs;
evs.read(ifile , number_events);
idxSesion.build(evs, 2);
idxSesion = rawFilterIndex(evs, idxSesion, "Type", "purchase");
```

Ahora nuestro índice apunta solo a los elementos de tipo "purchase", ordenados por sesión, como se muestra en el siguiente listado:

A partir de esa información podemos calcular fácilmente el conjunto de relaciones, pero debemos de tener cuidado de NO considerar aquellos eventos que no están asociados a una marca, esto es, los que tienen la cadena vacía como valor de marca. Así, dada una sesión `lower_bound` y `upper_bound` nos dará el rango de eventos que pertenecen a esa sesión (donde como hemos dicho podemos no considerar los que no tienen



idxsesion

	key,	pos
	S1,	0
	S1,	1
	S1,	2
	S1,	3
	S1,	4
	S1,	7
	S1,	9
	S2,	5
	S2,	6
	S2,	8
	S2,	14
	S3,	10
	S3,	11
	S3,	12
	S3,	13
	S4,	15
	S4,	17
	S5,	16
	S5,	18

idxsesion

key,	pos
S1,	4
S1,	7
S2,	6
S2,	8
S2,	14
S3,	12
S3,	13
S4,	15
S4,	17
S5,	16
S5,	18

evs::idxsession

	user,	type,	brand,	session
4	u1,	purchase,	A,	S1
7	u1,	purchase,	B,	S1
6	u2,	purchase,	B,	S2
8	u2,	purchase,	C,	S2
14	u2,	purchase,	D,	S2
12	u3,	purchase,	B,	S3
13	u3,	purchase,	A,	S3
15	u4,	purchase,	E,	S4
17	u4,	purchase,	,	S4
16	u1,	purchase,	B,	S5
18	u1,	purchase,	D,	S5

Figura 3: Índices y EventSet filtrados

marca asociada).

Siguiendo con nuestro ejemplo, podemos identificar el siguiente conjunto de relaciones:

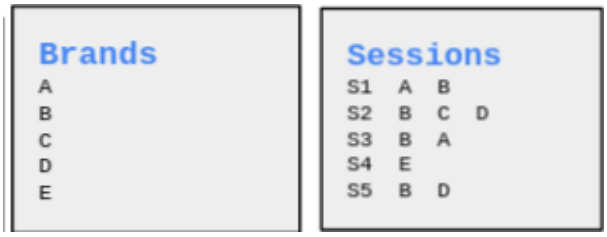


Figura 4: Valores únicos de marcas en total y por sesión

$$R(A, B, purchase)$$
$$R(B, C, purchase), R(B, D, purchase), R(C, D, purchase)$$
$$R(B, A, purchase)$$
$$R(B, D, purchase)$$

Es importante notar que la marca E no se encuentra relacionada con ninguna otra (hay dos eventos purchase en la sesión 4, pero uno es de marca desconocida). Por otro lado, fijémonos en que un análisis de las compras del usuario $u1$ no se puede considerar que las marcas A y D estén relacionadas ya que dichas comprar se realizaron en dos sesiones distintas.

Una vez que tenemos definida qué es una relación y cómo calcularlas, podremos crear el `grafo de marcas` donde los nodos son las distintas marcas que aparecen en el eventSet (incluyendo aquellas que no estén relacionadas) y el conjunto de relaciones que hemos calculado representan los enlaces del grafo, esto es, se incluye un arco no dirigido entre cada par de marcas que se encuentran relacionadas, como indica la siguiente figura.

Por ejemplo, si consideramos todas las marcas de nuestro conjunto de eventos podremos obtener un grafo[5] como el de la Figura 5. Este grafo será la base para calcular el PageRank.

3.3. Calculando el PageRank

A partir de este grafo podemos iniciar un proceso iterativo que nos llevará a calcular el valor del PageRank para cada nodo del mismo[6]. Por

[5]Visualización del grafo obtenido con la aplicación VOSviewer `https://www.vosviewer.com/`

[6]En la literatura podremos encontrar distintas formas para calcular el PageRank, pero la que nosotros utilizaremos nos permitirá profundizar en el uso de matrices bidimensionales dinámicas

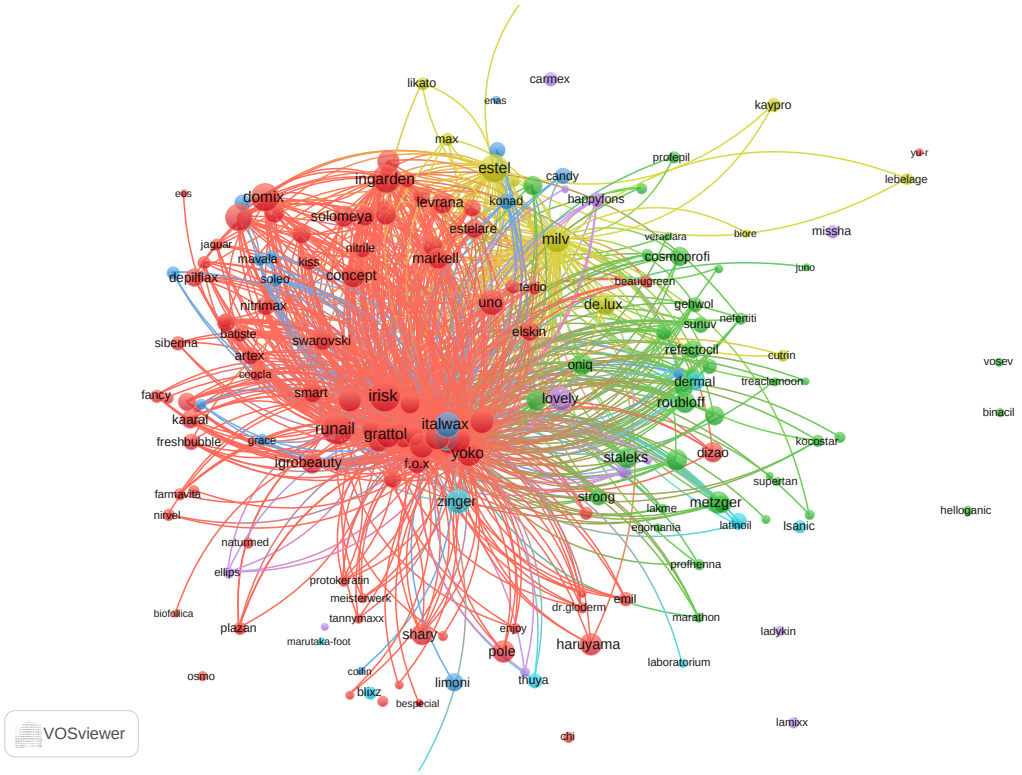


Figura 5: Grafo de Marcas

simplificar la notación hablaremos de $R(X, Y)$ y no de $R(X, Y, tipo)$, pues este ya lo podemos considerar implícito.

Para calcular el PageRank no trabajamos directamente sobre el 'grafo' sino que debemos de representarlo (a su conjunto de relaciones) mediante su matriz de adyacencia, M, sobre la que realizaremos un conjunto de transformaciones que nos garanticen la viabilidad del cálculo del valor de PageRank para cada nodo.

Para ello debemos de seguir los siguientes pasos:

Paso 1: Construir la matriz de adyacencia

La matriz de adyacencia será una matriz bidimensional de tamaño $n \times n$ siendo n el número de nodos en el grafo, que recordemos en nuestro caso se corresponde con el número de marcas. Así, si la fila i-ésima se corresponde con la marca A y la fila j-ésima se corresponde con la marca B, entonces $M[i][j] = k$ siendo k el número de veces en que la relación $R(A, B)$ es cierta. En caso de que nunca se dé dicha relación $M[i][j] = 0$. Como la relación $R()$ es simétrica la matriz resultante también lo será.

Matriz de Adyacencia Los valores de cada una de las celdas en la matriz de adyacencia representan el número de veces, k, que ocurre la relación entre dos marcas $brand\_i$ y $brand\_j$ en el dataset. Es una relación simétrica y por tanto

$$M[i][j] = M[j][i] = k, \text{ con } k >= 0 \text{ e } i \neq j$$

$$M[i][i] = 0, \text{ por definición}$$

Siguiendo con nuestro ejemplo, donde consideramos el conjunto de relaciones $R(A,B), R(B,C), R(B,D), R(C,D)$ podremos crear la siguiente matriz de adyacencia M (Tabla 1) donde se asume que la marca A está en la fila 0, la B en la fila 1, etc.

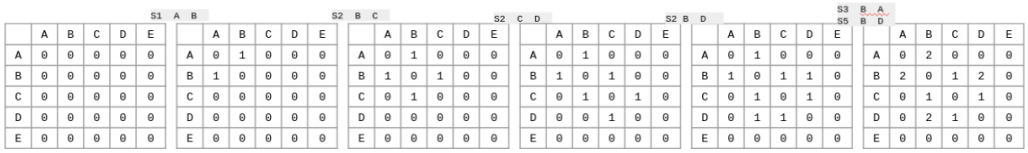


Figura 6: La matriz de adyacencia paso a paso

Cuadro 1: Representación matricial del conjunto de relaciones

M

0	2	0	0	0
2	0	1	2	0
0	1	0	1	0
0	2	1	0	0
0	0	0	0	0

nodo	fila/col
A	0
B	1
C	2
D	3
E	4

Paso 2: Normalización de la matriz

No entraremos en detalles sobre las razones que motivan a dar este paso, pero el objetivo final es normalizar cada una de las filas de modo que la suma de todos sus valores sea uno.

Para el lector más interesado, podemos interpretar estos valores de la siguiente forma: dado un nodo n_i, asociado a la fila i-ésima de la matriz, cada uno de los valores de dicha fila, $M[i][j]$ representa la probabilidad de que estando en el nodo n_i pasemos al nodo n_j.

$$M[i][j] = \frac{M[i][j]}{\sum_{k=0}^{COL} M[i][k]}$$

Para realizar este paso, primero deberemos de calcular cuánto es la suma de la fila y dividir cada valor por dicha cantidad. Hay que distinguir el caso en que un nodo no está conectado con ningún otro, como el nodo E. En esta situación no podemos seguir el algoritmo anterior (dividimos por cero), y en su lugar lo que se hace es asignar el valor $1/n$ a cada uno de los elementos de la fila, siendo n el número total de nodos.

Tras aplicar el segundo paso obtenemos la siguiente matriz (Tabla 2)

Cuadro 2: Matriz normalizada

M				
0	1	0	0	0
0,4	0	0,2	0,4	0
0	0,5	0	0,5	0
0	0,66	0,33	0	0
0,2	0,2	0,2	0,2	0,2

nodo	fila/col
A	0
B	1
C	2
D	3
E	4

Paso 3: Teletransporte

La idea que hay tras este paso es garantizar que desde un nodo cualquiera del grafo se puede alcanzar cualquier otro nodo, aunque con una probabilidad baja, de ahí la idea de teletransporte (es posible alcanzar cualquier nodo aunque no exista relación directa). Para ello, se considera una valor α con $0 \leq \alpha \leq 1$ de forma que con probabilidad α se siguen los enlaces del grafo y con probabilidad $1-\alpha$ se hace el teletransporte de forma aleatoria a cualquier otro nodo del grafo. Se dice que el valor de α utilizado por Google es de $0{,}85$.

Para permitir esto, la matriz normalizada se modifica de la siguiente forma

$$M[i][j] = \alpha * M[i][j] + \frac{(1-\alpha)}{n}$$

En nuestro ejemplo, y asumiendo $\alpha = 0{,}85$ obtenemos la siguiente matriz (Tabla 3)

Cuadro 3: Matriz con Teletransporte, $\alpha = 0{,}85$

M				
0,03	0,88	0,03	0,03	0,03
0,37	0,03	0,2	0,37	0,03
0,03	0,455	0,03	0,455	0,03
0,03	0,596	0,313	0,03	0,03
0,2	0,2	0,2	0,2	0,2

nodo	fila/col
A	0
B	1
C	2
D	3
E	4

Paso 4: Cálculo de PageRank

Ya lo tenemos todo disponible para poder realizar los cálculos de PageRank. Como hemos dicho, el resultado final del algoritmo será para cada nodo n_i del grafo (para cada marca) su valor de PageRank, $PR(n_i)$. La pregunta que nos podemos hacer en este momento es cómo almacenamos los distintos valores que nos devuelve PageRank. Una posible solución en nuestro contexto será crearnos una matriz bidimensional con una fila y n columnas, de modo que en la columna j-ésima almacenaremos el valor de PageRank del nodo j-ésimo del grafo (fila j de la matriz).

Ahora ya tenemos todos los elementos para poder ejecutar el algoritmo. Como hemos dicho es un algoritmo iterativo donde en cada paso se va refinando el valor del PageRank de cada nodo. Inicialmente, se asigna un valor aleatorio de PageRank, aunque impondremos como requisito que la suma de todos los valores de PageRank sea igual a uno.

En cada paso del algoritmo se actualiza el valor de PR al realizar la multiplicación de PR por la matriz M, esto es $PR_{t+1} = PR_t * M$. Por ejemplo, partiendo de un valor inicial es $PR_0(n_i) = 1/n, \forall i \in 0 \ldots, n-1$, tendremos que $PR_1 = PR_0 \times M$ como se indica el la Tabla 4.

Cuadro 4: Cálculo de PR_1

PR_0

	A	B	C	D	E
	0,2	0,2	0,2	0,2	0,2

X

0,03	0,88	0,03	0,03	0,03
0,37	0,03	0,2	0,37	0,03
0,03	0,455	0,03	0,455	0,03
0,03	0,596	0,313	0,03	0,03
0,2	0,2	0,2	0,2	0,2

=

PR_1

	A	B	C	D	E
	1,32	0,432	0,154	0,217	0,064

El proceso se repite, esto es, $PR_2 = PR_1 \times M, PR_3 = PR_2 \times M, \ldots$ hasta llegar a una situación de estabilidad, es decir, el resultado converge a unos valores. Esta situación se alcanza cuando los cambios en los valores de PR sean mínimos:

$$\sum_{i=0}^{n-1} |PR_{t+1}(i) - PR_t(i)| < umbral$$

aunque también se puede detener cuando se hayan alcanzado un número determinado de iteraciones.

Mostramos las 5 primeras iteraciones del PageRank sobre nuestro ejemplo, así como el punto donde se alcanza la estabilidad (la diferencia entre los valores de PageRank entre iteración es inferior que el umbral definido en este caso a 0.0001)

```
          A     B     C     D     E
it  0:  0.2   0.2   0.2   0.2   0.2
it  1: 0.132 0.432 0.154 0.217 0.064
it  2: 0.187 0.341 0.175 0.253 0.040
it  3: 0.153 0.415 0.166 0.227 0.036
it  4: 0.177 0.366 0.171 0.248 0.036
it  5: 0.160 0.400 0.168 0.233 0.036
...
it 16: 0.167 0.386 0.169 0.239 0.036
```

Por tanto, un esquema del algoritmo PageRank sería el siguiente:

```
Algoritmo PageRank(Matriz & M, int it_max, double umbral)
1.- Inicializar PR
      PR_ant = 1/n_nodos
      iter = 0
2.- Mientras (iter < max_iter && !estable){
      PR_nuevo = PR_ant * M
      valor = suma de valores absolutos de (PR_nuevo - PR_ant)
      if (valor < umbral)
         estable = true
      PR_ant = PR_nuevo
      iter++
    }
3.- Devuelve PR_nuevo
```

En el siguiente gráfico, Figura 7 se muestra los nodos con mayores valores de PageRank para nuestra tienda de comercio electrónico (visualización de densidad por PageRank).

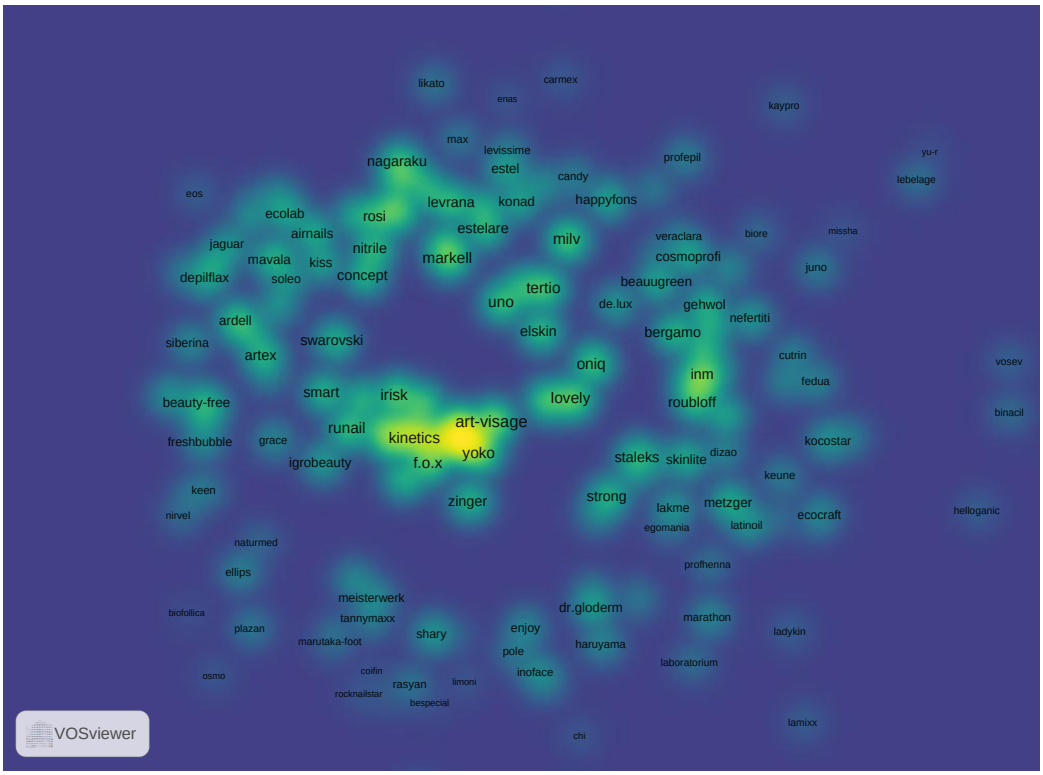


Figura 7: Grafo de Desidad

4. Desarrollo del Sistema de Recomendación para nuestra tienda de comercio electrónico

Una vez que hemos visto el algoritmo PageRank y cómo se calcula, la pregunta que nos hacemos es cómo podemos integrar todo lo anterior en nuestra tienda de comercio electrónico.

En este caso, nos planteamos una situación en la que un cliente se ha autentificado en la tienda y tenemos como objetivo mejorar su experiencia ofreciéndole un total de, por ejemplo, 10 marcas (productos en general) en las que puede estar interesado. Como hemos dicho, esta recomendación se obtendría al considerar las 10 marcas con mayor valor de PageRank.

Calcular el PageRank es un proceso costoso y se puede almacenar en disco una vez calculado.

Cuadro 5: Recomendaciones

Sugerencias				
isrisk	ingarden	domix	estel	masura
milv	italwax	grattol	bpw.style	kapous

4.1. Nuestro objetivo

Para poder dar solución a nuestro problema planteamos la implementación de una clase capaz de representar matrices algebraicas con etiquetas y el programa principal que será donde se realizará el cómputo de los valores del PageRank, así como la selección de las mejores marcas, esto es, la recomendación propiamente dicha.

Para ello, los parámetros de entrada para nuestro programa serán:

- el nombre de fichero que contenga los eventos de nuestro comercio durante un periodo amplio de tiempo. Este es el fichero que utilizaremos para crear el grafo de relaciones.
- Un entero que represente cuántos eventos leer.
- Un string representando el tipo de relación sobre el que calcular las relaciones.
- Un entero que represente cuántas recomendaciones obtener.
- El modo en el que queremos presentar las recomendaciones.
- El nombre de un fichero de salida para guardar los valores del PageRank.

Los parámetros tales como α, el umbral de convergencia y el número máximo de iteraciones se encuentran definidos por defecto en el programa principal.

En concreto, nuestra llamada sería como se indica:

```
Shopping5 -input <filename> -events <number_events> -K <number>
        [-type <purchase|view|cart|remove_from_cart> -display <Fancy|Framed|Plain> -output <ofilename>]

-input <filename>
      Read events from dataset in <filename>
-events <number_events>
      Maximum number of events to read from dataset filename
-K <number>
      Number of recommendations
-type <purchase|view|cart|remove_from_cart>
      Used to define the relationships (default purchase)
-display <Fancy|Framed|Plain>
      Display mode (default Plain)
-output <ofilename>
      If no file is given ofilename =ifilename+\"pgrk\"
```

Nuestro programa deberá obtener como salida el conjunto de recomendaciones que debemos presentar a cada uno de los usuarios que actualmente están activos. Para ello, seguiremos un esquema **parecido** al que se indica

```
- Procesar los argumentos de entrada del programa

- Leer el fichero y construir el índice para seleccionar las marcas (brand) que sean adecuadas para la
      recomendación, i.e. aquellas que se van incluir como filas/columnas en la matriz de adyacencia

    - lectura del EventSet
    - construir índice por sesión
    - filtrado por los eventos que sean de interés
    - identificar el conjunto de brands único (serán las etiquetas de la matriz de adyacencia), eliminando
      la marca desconocida ("")

- Crear la matriz de adyacencia, M
- Calcular el teletransporte
       M = alpha * M[i][j] + (1-alpha)/N

- Calcular el PageRank y guardarlo en disco (para posibles futuros usos)
- Calcular las topK recomendaciones y mostrarlas al usuario
```

En el fichero asociado al programa principal tenemos comentadas todas las funciones que debemos implementar para alcanzar los objetivos, además de la clase Matrix que implementaremos como una matriz 2D capaz de representar matrices algebraicas etiquetadas.

5. Matriz Bidimensional Etiquetada: Implementación versión 2D

Esta clase nos permitirá representar una matriz bidimensional clásica (sin etiquetas) como la del siguiente ejemplo, Tabla 6, donde el acceso se hace con las posiciones enteras de fila y columna (empezando en cero). Así, el valor en la intersección de la fila 1 - columna 1 será de 8 y el valor en fila 0 - columna 5 será de 14.

Cuadro 6: Matriz No etiquetada

2	7	11	12	13	14
3	8	12	15	16	17
4	9	13	16	18	19
5	10	14	17	19	20
0	1	2	3	4	5
1	6	7	8	9	10

Pero también nos permitirá representar una matriz bidimensional con etiquetas tanto en filas como en columnas, como la que tenemos en la Tabla 7, donde el acceso a las distintas posiciones de la misma se puede hacer de forma dual, esto es, utilizando la etiqueta de la fila y la etiqueta de la columna, por ejemplo el valor de la matriz en las posiciones dadas por la fila “V”, columna “D”, se corresponde con el 6 y el valor en intersección “U”-“F” es 5. De forma análoga podremos acceder a dichos valores utilizando las posiciones de fila y columna, fila 5 columna 1 tiene el valor de 6, y fila 4 columna 5 tiene valor de 5.

Cuadro 7: Matriz Etiquetada

	A	D	E	B	C	F
W	2	7	11	12	13	14
X	3	8	12	15	16	17
Y	4	9	13	16	18	19
Z	5	10	14	17	19	20
U	0	1	2	3	4	5
V	1	6	7	8	9	10

5.1. Representación

Al la hora de representar la matriz, debemos separar la gestión de los valores de las celdas propiamente dichas, que se corresponde con la

implementación de una matriz bidimensional donde cada fila se aloja en un bloque independiente de datos (del tamaño del número de columnas), de la gestión de las etiquetas en filas y columnas. Para ello, proponemos el uso de la siguiente estructura:

```
class matrix {
private:

    double ** _data;  // puntero a inicio bloque de datos
    Index _rowLabels; // índice con las etiquetas de la fila
    Index _colLabels; // índice con las etiquetas de la columna

    size_t _nrows; // número de filas
    size_t _ncols; // número de columnas
public:
...
};
```

En esta estructura `_data` se representa la matriz propiamente dicha, sus celdas, implementada siguiendo los patrones de la matriz 2D dada en teoría, por lo que no se detallarán en esta memoria. En caso de duda consultar al profesor.

También utilizaremos dos objetos de tipo índice, Index, entendido como un conjunto de pares `<string,int>` donde el campo string hace referencia a la etiqueta (nombre) de fila/columna y el `int` hace referencia a la posición en la matriz en la que se encuentran almacenados los datos de dicha fila/columna. Obviamente, debemos de garantizar que no existan etiquetas repetidas dentro del conjunto `_rowLabels` (de forma similar para `_colLabels`). Para determinar si una matriz es etiquetada o no es suficiente con consultar el tamaño (size) de `_rowLabels` y `_colLabels` (si alguno es mayor que cero, la matriz será etiquetada).

En el siguiente esquema presentamos lo que sería la representación de la matriz de la Tabla 6:

Cuadro 8: Representación de Matriz Etiquetada

	data						EtqCols	Pos	EtqFilas	Pos
	2	7	11	12	13	14	A	0	U	4
_etiq=true	3	8	12	15	16	17	B	3	V	5
_filas = 6	4	9	13	16	18	19	C	4	W	0
_cols=6	5	10	14	17	19	20	D	1	X	1
	0	1	2	3	4	5	E	2	Y	2
	1	6	7	8	9	10	F	5	Z	3

5.2. Especificación y ejemplo de uso

Especificación Presentamos someramente los métodos a implementar, la documentación detallada se encuentra en el fichero Matrix.h

```
/** Default constructor */
Matrix();

/** Copy constructor, */
Matrix(const Matrix& orig);

/** Creates a 2D unlabeled matrix with size r x c, being all its setted to the defValue */
Matrix(size_t r, size_t c, double defValue = 0.0);

/** Creates a 2D labeled matrix with size nrows x ncols */
Matrix(const std::string *rowLabels, size_t nrows, const std::string * colsLabels, size_t ncols,
double defValue = 0.0);
```

```
    /** Destructor */
    virtual ~Matrix();

    /*********************************************/
    /****** Consultors ***************************/
    /*********************************************/

    /** Number of columns */
    size_t columns()const;
    /** Number of rows */
    size_t rows() const;
    /** Element at M[row][col]. */
    const double & at(size_t row, size_t col) const;
    /** Element at M[row][col]. */
    const double & operator()(size_t row, size_t col) const;
    /** Element at M[rowLabel][colLabel]. */
    const double & at(const std::string & rowLabel, const std::string & colLabel) const;
    /** Element at M[rowLabel][colLabel]. */
    const double & operator()(const std::string & rowLabel, const std::string & colLabel) const;
    /** Gives the label associated to the row in the matrix */
    std::string labelAtRow(size_t row) const;
    /** Gives the label associated to the column col in the matrix */
    std::string labelAtCol(size_t col) const;
    /** Given a label, return its associated row */
    int getRowLabels(std::string *) const;
    /** Given a label, return its associated column */
    int getColLabels(std::string *) const;
    /** Return the position of the row associated to the label. */
    int rowIndexOf(const std::string & label) const;
    /** Return the position of the col associated to the label. */
    int colIndexOf(const std::string & label) const;

    /*********************************************/
    /******************** MODIFIERS **************/
    /*********************************************/

    /** Element at M[row][col]. Check is done */
    double & at(size_t row, size_t col);
    /** Element at M[row][col]. No Check is done */
    double & operator()(size_t row, size_t col);
    /** Element at M[rowLabel][colLabel]. Check is done */
    double & at(const std::string & rowLabel, const std::string & colLabel);
    /** Element at M[rowLabel][colLabel]. No check is done */
    double & operator()(const std::string & rowLabel, const std::string & colLabel);
    /** Change the size of the matrix to nrows x ncols initializing all its value to defValue */
    void resize(size_t nrows, size_t ncols, double defValue = 0.0);
    /** Change the size of the matrix to nrows x ncols initializing all its value to vdef. The rows/
columns are labeled with rowLabels/colLabels */
    void resize(const std::string * rowLabels, size_t nrows, const std::string * colLabels, size_t ncols,
   double vdef = 0);
    /** Set the matrix to the default values */
    void clear();
    /** Modify the values in the matrix in such a way that for each row, the sum of all its elements is
1.0 */
    Matrix & normalize();

    /*********************************************/
    /***************** OPERATORS *****************/
    /*********************************************/

    /** Assign new contents to the matrix, replacing its current contents, and modifying its size
accordingly. */
    Matrix & operator=(const Matrix & m);
    /** Multiply each element in the matrix by val */
    Matrix & operator*=(double val);
    /** Add a constant value to each element in the matrix */
    Matrix & operator+=(double val);
    /** Matrix multiplication. */
    Matrix operator*(const Matrix & m) const;
    /** Multiply the matrix by a constant val */
    Matrix operator*(double val) const;
    /** Add val to each element in the matrix*/
    Matrix operator+(double val) const;

    /*********************************************/
    /************* INPUT / OUTPUT ****************/
    /*********************************************/

    /** Save the matrix in a file */
    void save(const std::string & filename) const;
    /** Load a matrix from the file filename */
    void load(const std::string & filename);

    /*********************************************/
    /*********** FRIEND METHODS/OPERATORS ********/
    /*********************************************/

   /** Return a new matrix obtained by computing: ival + M */
   Matrix operator+(double ival, const Matrix & M);
   /** Return a new matrix obtained by computing: ival * M */
   Matrix operator*(double ival, const Matrix & M);
   /** Insert the data representing the matrix into the ostream */
   std::ostream & operator<<(std::ostream & os, const Matrix & m);
```

Ejemplos de uso: Supongamos que queremos crear la siguiente matriz (Tabla 9) de datos:

Cuadro 9: Ejemplo de matriz no etiquetada

0	1	2	3	4	5
1	6	7	8	9	10
2	7	11	12	13	14
3	8	12	15	16	17
4	9	13	16	18	19
5	10	14	17	19	20

La función `asignaValores` nos permite inicializar la matriz con los datos deseados

```
/** Creamos la matriz de la Tabla 9. Matriz Ejemplo ***/
void asignaValores(matrix & m, int f, int c){
    double v = 0;
    m.resize(f,c);
    for (int i=0; i<f; i++)
        for (int j=i; j<c;j++){
            m.at(i,j) = v;  // asignamos valor con at
            m(j,i) = v; // asignamos valor con el operator()
            v=v+1;
        }
}

/*** Ejemplo de creación de una matriz etiquetada ***/
Matrix MatrizEjemploEtiquetada() {
    string et[5] = {"a", "b", "c", "d", "e"};
    Matrix ej(et, 5, et, 5);
    ej(0, 1) = 1;
    ej("b", "a") = ej("b", "c") = 1;
    ej("b", "d") = 1;
    ej(2, 1) = ej(2, 3) = 1;
    ej(3, 1) = ej(3, 2) = 1;
    return ej;
}
```

En el siguiente código presentamos distintas alternativas para enviar los elementos de la matriz a un flujo de salida

```
int main(){
  matrix m;
  asignaValores(m,6,6);
  ...
  cout << m << endl; // mostramos la matriz usando el operador <<

  string filas[6]={ "A","D","E","B","C","F"};
  string cols[6] = { "W", "X","Y","Z","U","V"};

  matrix etq(filas,6,cols,6,0);
  asignaValores(etq,6,6);

  cout << m<< endl;
  cerr << m << endl;

  m.normalize();

  matriz final;
  final = etq*2 + 5;
  final = final * etq;
  cout << final << endl;

  // Salvamos matriz en disco
  m.save("ficheroSalida.txt");  // la guardamos como texto en ficheroSalida.txt

}
```

6. Ejemplo de salida paso a paso de nuestro programa

En este caso, asumimos que queremos calcular el valor de PageRank considerando relaciones de tipo “cart” sobre marcas (R(X,Y,cart)) considerando como entrada el fichero de datos `./data/ECommerce162.dataset`. Los parámetros concretos de entrada han sido:

```
-input ./data/ECommerce162.dataset -K 5 -type cart -events 10000 -display Framed
```

Si seguimos los pasos indicados en este esquema

```
- Procesar los argumentos de entrada del programa

- Leer el fichero y construir el índice para seleccionar las marcas (brand) que sean adecuadas para la
    recomendación, i.e. aquellas que se van incluir como filas/columnas en la matriz de adyacencia

   - lectura del EventSet
   - construir índice por sesión
   - filtrado por los eventos que sean de interés
   - identificar el conjunto de brands único (serán las etiquetas de la matriz de adyacencia), eliminando
     la marca desconocida ("")

- Crear la matriz de adyacencia, M
- Calcular el teletransporte
      M = alpha * M[i][j] + (1-alpha)/N

- Calcular el PageRank y guardarlo en disco (para posibles futuros usos)
- Calcular las topK recomendaciones y mostrarlas al usuario
```

podemos obtener como resultado la siguiente salida

```
Reading from ./data/ECommerce162.dataset

Number of events with non null session in index: 162
After filtering sessions (Type == cart): 51 sessions

There are 11 unique brands in the filtered sessions

Adjacency Matrix
11 11
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0 2 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0
0 0 0 12 3 1 2 2 1 1 0
0 0 12 0 36 12 24 24 12 12 0
0 0 3 36 0 3 6 6 3 3 0
0 0 1 12 3 0 2 2 1 1 0
0 0 2 24 6 2 0 4 2 2 0
0 0 2 24 6 2 4 0 2 2 0
0 0 1 12 3 1 2 2 0 1 0
0 0 1 12 3 1 2 2 1 0 0
0 0 0 0 0 0 0 0 0 0 0

After teletranspotation
11 11
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0136364 0.863636 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364
0.863636 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364 0.0136364
0.0136364 0.0136364 0.0136364 0.477273 0.129545 0.0522727 0.0909091 0.0909091 0.0522727 0.0522727 0.0136364
0.0136364 0.0136364 0.0909091 0.0136364 0.245455 0.0909091 0.168182 0.168182 0.0909091 0.0909091 0.0136364
0.0136364 0.0136364 0.0561364 0.523636 0.0136364 0.0561364 0.0986364 0.0986364 0.0561364 0.0561364 0.0136364
0.0136364 0.0136364 0.0522727 0.477273 0.129545 0.0136364 0.0909091 0.0909091 0.0522727 0.0522727 0.0136364
0.0136364 0.0136364 0.0541126 0.499351 0.135065 0.0541126 0.0136364 0.0945887 0.0541126 0.0541126 0.0136364
0.0136364 0.0136364 0.0541126 0.499351 0.135065 0.0541126 0.0945887 0.0136364 0.0541126 0.0541126 0.0136364
0.0136364 0.0136364 0.0522727 0.477273 0.129545 0.0522727 0.0909091 0.0909091 0.0136364 0.0522727 0.0136364
0.0136364 0.0136364 0.0522727 0.477273 0.129545 0.0522727 0.0909091 0.0909091 0.0522727 0.0136364 0.0136364
0.090909  0.090909  0.090909  0.090909 0.090909 0.090909  0.090909  0.090909  0.090909  0.090909  0.090909

PageRank at iteration 0
1 11
pr
bpw.style cosmoprofi domix estel freedecor haruyama ingarden irisk italwax roubloff runail
0.090909  0.090909  0.090909  0.090909 0.090909 0.090909  0.090909  0.090909  0.090909  0.090909  0.090909

PageRank at iteration 1
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0979339 0.0979339 0.0494461 0.323932 0.105962 0.0494461 0.0778965 0.0778965 0.0494461 0.0494461 0.0206612

PageRank at iteration 2
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0984767 0.0984767 0.0568045 0.236645 0.132169 0.0568045 0.0958911 0.0958911 0.0568045 0.0568045 0.0152329
```

```
PageRank at iteration 3
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985187 0.0985187 0.0530636 0.280717 0.119296 0.0530636 0.0879405 0.0879405 0.0530636 0.0530636 0.0148135

PageRank at iteration 4
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985219 0.0985219 0.0548125 0.259459 0.125816 0.0548125 0.0918253 0.0918253 0.0548125 0.0548125 0.014781

PageRank at iteration 5
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985221 0.0985221 0.0539615 0.269798 0.122639 0.0539615 0.0899466 0.0899466 0.0539615 0.0539615 0.0147785

PageRank at iteration 6
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985222 0.0985222 0.0543746 0.264775 0.124185 0.0543746 0.0908592 0.0908592 0.0543746 0.0543746 0.0147783

PageRank at iteration 7
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985222 0.0985222 0.0541739 0.267216 0.123434 0.0541739 0.0904158 0.0904158 0.0541739 0.0541739 0.0147783

PageRank at iteration 8
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985222 0.0985222 0.0542714 0.26603 0.123799 0.0542714 0.0906313 0.0906313 0.0542714 0.0542714 0.0147783

PageRank at iteration 9
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985222 0.0985222 0.054224 0.266606 0.123622 0.054224 0.0905266 0.0905266 0.054224 0.054224 0.0147783

PageRank at iteration 10
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985222 0.0985222 0.0542471 0.266326 0.123708 0.0542471 0.0905775 0.0905775 0.0542471 0.0542471 0.0147783

PageRank at iteration 11
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985222 0.0985222 0.0542359 0.266462 0.123666 0.0542359 0.0905528 0.0905528 0.0542359 0.0542359 0.0147783

PageRank at iteration 12
1 11
pr
italwax estel roubloff bpw.style freedecor domix ingarden irisk cosmoprofi runail haruyama
0.0985222 0.0985222 0.0542413 0.266396 0.123686 0.0542413 0.0905648 0.0905648 0.0542413 0.0542413 0.0147783

Convergence reached in 13 iterations (stability last value = 6.42734e-05)

First 10 stability values
     0.524      0.185      0.088      0.043      0.021      0.010      0.005      0.002      0.001      0.001
     0.880

      it 0       it 1       it 2       it 3       it 4       it 5       it 6       it 7       it 8       it 9


??????????????????????????????????????????????????????????
?    0.0985?    0.0985?    0.0906?    0.2664?    0.1237?    0.6777
??????????????????????????????????????????????????????????
?   italwax?     estel?  ingarden? bpw.style? freedecor?
??????????????????????????????????????????????????????????
```

7. Shopping5, práctica a entregar

Como hemos comentado, junto con esta documentación se os entregan los siguientes ficheros: `matrix.h, matrix.cpp, main.cpp`, El programa principal es el encargado de realizar las recomendaciones y la clase `Matrix` es la estructura utilizada de forma interna para realizar los cómputos. Se os recomienda hacer una copia del proyecto (llamarla Shopping5) e incluir en el mismo la clase `Matrix` así como cambiar el fichero `main.cpp` anterior por este main.

Indicaros que para facilitaros la tarea os hemos dado implementados algunos métodos, tanto del programa principal como de la clase `Matrix`. Todas las clases anteriores seguirán como en la práctica de shopping 4 a excepción de los métodos `build`, `setIOnWhich` y `rawFilterIndex` de Index, en el que habrá que considerar si el índice se construye sobre marcas, usuarios o sesiones. Para ello basta modificar en el método `build` la línea:

```
field = onBrand ? "Brand" : "UserID";
```

Por lo siguiente:

```
if (onBrand == 2)
  field = "Session";
else if (onBrand == 0)
  field = "UserID";
else
  field = "Brand";
```

De forma similar, se deberá modificar `setIOnWhich` para considerar los tres posibles valores para `_onBrand`. Para el método `rawFilterIndex` se deberá cambiar la condición del `if` añadiendo también la condición:

```
|| (field=="Session" && indx.getIOnWhich()==2)
```

A diferencia de las prácticas anteriores NO se entregan un conjunto de tests, por lo que sois vosotros los que debéis probar las distintos métodos para aseguraros de su correcto funcionamiento. Así, se recomienda que cuando implementeis un método lo probéis desde el propio main chequeando que los resultados que se obtienen son los esperados (bien utilizando el debugger, opción que en la mayoría de los casos es la mejor, o bien haciendo couts en el terminal). En cualquier caso, os entregamos también algunos conjuntos de datos con su salida en la que se ilustra el funcionamiento del algoritmo en los distintos pasos (os servirán para chequear que todo es correcto).

Al igual que en la práctica anterior se recomienda el uso de la herramienta **valgrind** para ver si hay algún tipo de error/fuga de memoria. Para ello puede consultarse el guión sobre valgrind puesto a disposición en Prado.

7.1. Entrega de la práctica

Una vez terminada la práctica se debe hacer un zip (se sugiere utilizar la script `runZipProject.sh`) excluyendo las carpetas `./dist/`, `./build/`, `./nbproject/private/`, `./doc/html/` y `./dos/latex/` y subirla a Prado antes de la fecha de cierre de la entrega.