Explicación de Implementación

1. ***LINKS:***
   a. ***Base de Datos Preprocesada:*** ***https://drive.google.com/drive/folders/1G7y2n5Hz7w_DgzQQJxaRBtGVhMSQonfF?usp=sharing***
   b. ***Github:*** ***https://github.com/Kath17/Recommendation_System***
   c. ***Reporte Técnico:***

      https://es.overleaf.com/read/ysgzphykqkhs

2. ***ALMACENAMIENTO Y LEVANTAMIENTO DE LA BASE DE DATOS:***

   Almacenamiento:

   Para el manejo de la base de datos se usó el contenedor ***map*** de la librería estándar ***STL*** de C++. ***Map*** es un contenedor donde cada elemento tiene una llave y un elemento mapeado. Dos elementos no pueden ser mapeados por la misma llave.
   Aprovechando estas características se diseñó una estructura que represente a una matriz. Usando un ***map*** de ***map***, donde la llave del primer ***map*** guarda el nombre de los usuarios y el segundo componente guarda otro ***map*** que contiene los ítems y la calificación de las películas que vio.

```
typedef string UserId;
typedef string ItemId;
typedef float RatingVal;

typedef map < UserId, map <ItemId, RatingVal> > db;
```

   Levantamiento de la BD:

   Para leer la Base de Datos estamos usando la clase fstream de la librería estándar STL, la cual nos permite acceder al archivo.
   Si el archivo no se encuentra, se imprime un mensaje de error.
   Al existir la Base de datos se lee línea por línea, guardandola en el string ***line***.

```
db getBD(string fileName, char delimit, int &ultimo_usuario){

    db BDatos;
    ifstream File(fileName.c_str());

    if (!File) {
    std::cout << "Error, could not open file." << std::endl;
    }

    vector <string> temp;
    temp.reserve(3);
    string line;
```

Cada línea es separada mediante su delimitador “,”, guardando cada parte en un vector llamado “**temp**”. Ejemplo:

line = 4598,”Titanic (1984)”,4.5
temp[0] = 4598
temp[1] = Titanic (1984)
temp[2] = 4.5

El valor de temp[2], que es el rating, es guardado como float en la variable **floatVar**, por lo cual se hace la conversión de string a float.

```
    while(getline(File,line))
    {
        stringstream  lineStream(line);

        temp = split_string_nos(line, delimit);

        string QString = temp[2];
        istringstream StrToFloat(QString );
        float floatVar;
        StrToFloat >> floatVar;

        BDatos[temp[0]][temp[1]] = floatVar;
    }
```

```
    string QString = temp[0];
    istringstream StrToInt(QString );
    int floatVar;
    StrToInt >> floatVar;

    ultimo_usuario = floatVar;

    File.close();
    return BDatos;
}
```

Finalmente se agrega cada línea dentro del map de map de la siguiente manera:

BDatos[temp[0]][temp[1]] = floatVar;

Las líneas de código luego del while sólo sirven para obtener el ID del último registro en la Base de datos.

3. *IMPLEMENTACIÓN K-NN*

El **K-nn** nos devuelve los k vecinos más cercanos de un usuario. En la implementación tenemos un método que se encarga de esto y se llama *vecino_cercano*. Este método nos devuelve un vector de duplas, donde el primer valor es la distancia de usuarios y el segundo valor es el usuario con el que se está comparando. Este método recibe 3 entradas: *user*, k, algoritmo, donde user es el usuario del cual se quieren calcular los vecinos más cercanos, k es el número de vecinos más cercanos que calculares y algoritmo es la métrica de distancia que se está usando.

Primero creamos un vector llamado *distanciasTodo* que contendrá las distancias con todos los vecinos. Luego se elige la métrica de distancia que se utilizará (Manhattan, Euclidean, Pearson, Similitud del Coseno). Posteriormente se recorre todos los usuarios cuidando que no se haga la comparación consigo mismo, todos los resultados de estas operaciones son almacenadas en *distanciasTodo*. Posteriormente se usa el ***sort*** de *STL* para ordenarlo en forma creciente si la métrica de distancia es de Manhattan o Euclidean y decreciente si la métrica de distancia es Correlación de Pearson o Similitud de Coseno. Finalmente se crea un vector *distancias* que contendrá la salida con los *K* vecinos más cercanos que son extraídos de *distanciasTodo* que ya es un vector ordenado dependiendo de la

métrica de distancia. Se hace una validación de si n es menor que K, donde n es el número de vecinos cercanos, el K se actualice con el valor de n.

```
vector<tuple<float,string>> vecino_cercano(string user,int k,string algoritmo)
{
    vector<tuple<float,string>> distanciasTodo;
    if(algoritmo == "Manhattan" || algoritmo == "Euclidean")
    {
        if (algoritmo == "Manhattan")
        {
            for(auto users: data)
            {
                if (users.first != user)
                    distanciasTodo.push_back(make_tuple(manhattan(user,users.first),users.first));
            }
        }
        else {
            for(auto users: data)
            {
                if (users.first != user)
                    distanciasTodo.push_back(make_tuple(euclidean(user,users.first),users.first));
            }
        }

        sort(distanciasTodo.begin(),distanciasTodo.end());

        vector<tuple<float,string>> distancias;
        distancias.reserve(k);
        int n = distanciasTodo.size();
        if (n < k)
            k = n;
        for(int i=0; i <k; i++)
            distancias.push_back(distanciasTodo[i]);
        return distancias;

    }
```

```
    else if (algoritmo == "Pearson" || algoritmo == "Similitud del Coseno")
    {
        if (algoritmo == "Pearson")
        {
            for(auto users: data)
            {
                if (users.first != user)
                    distanciasTodo.push_back(make_tuple(pearson(user,users.first),users.first));
            }
        }
        else {
            for(auto users: data)
            {
                if (users.first != user)
                    distanciasTodo.push_back(make_tuple(sim_coseno(user,users.first),users.first));
            }
        }

        sort(distanciasTodo.rbegin(),distanciasTodo.rend());
```

```
        vector<tuple<float,string>> distancias;
        distancias.reserve(k);
        int n = distanciasTodo.size();
        if (n < k)
            k = n;
        for(int i=0; i <k; i++)
            distancias.push_back(distanciasTodo[i]);
        return distancias;
    }
    else{
        cout<<"Ingrese una distancia valida"<<endl;
    }
}
```

4. LENGUAJE DE PROGRAMACIÓN, LIBRERÍAS Y QUÉ HACEN

- Lenguaje de programación: C++
    - En un primer momento se desarrollo con python, pero hubo problemas con el manejo de la data. Entonces se optó por C++ que es más rápido porque es compilado y no interpretado. Además de que nos ofrece **map** que es un contenedor que que nos ayuda en el momento de la construcción de la arquitectura.

- Librería: Librería estándar de STL:
    - #include <map>
        - Es un contenedor donde cada elemento tiene una llave y un elemento mapeado. Dos elementos no pueden ser mapeados por la misma llave.

    - #include <vector>
        - son los mismos que los arreglos dinámicos con la capacidad de redimensionarse automáticamente cuando se inserta o elimina un elemento, y su almacenamiento se maneja automáticamente por el contenedor

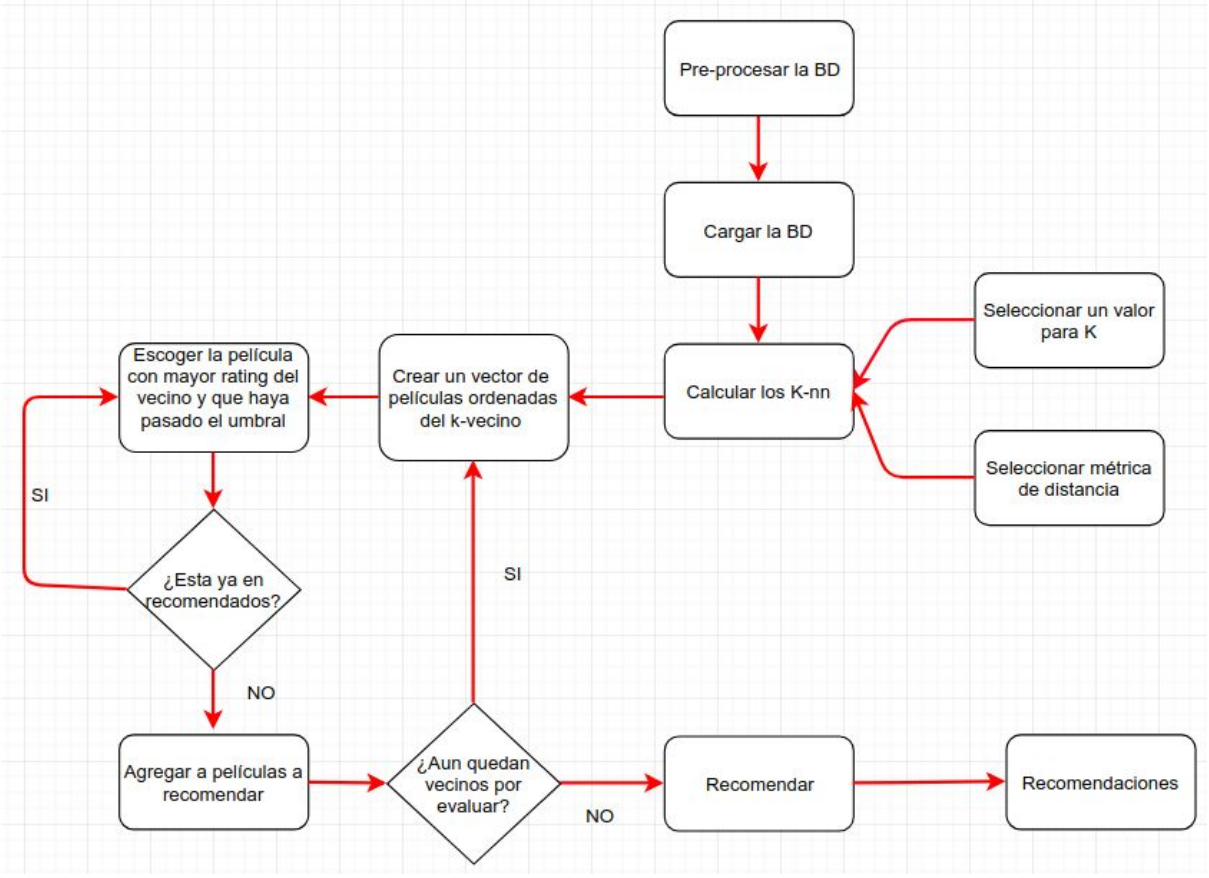
    - #include <tuple>
        - son objetos que agrupan elementos de posiblemente diferentes tipos en un solo objeto, al igual que los objetos de pares para pares de elementos, pero generalizados para cualquier número de elementos.

    - #include <algorithm>

- define una colección de funciones especialmente diseñadas para ser utilizadas en rangos de elementos.

- #include <fstream>
  - Clase de flujo de entrada / salida para operar en archivos.
- sort
  - Ordena los elementos en el rango [primero, último]en orden ascendente.

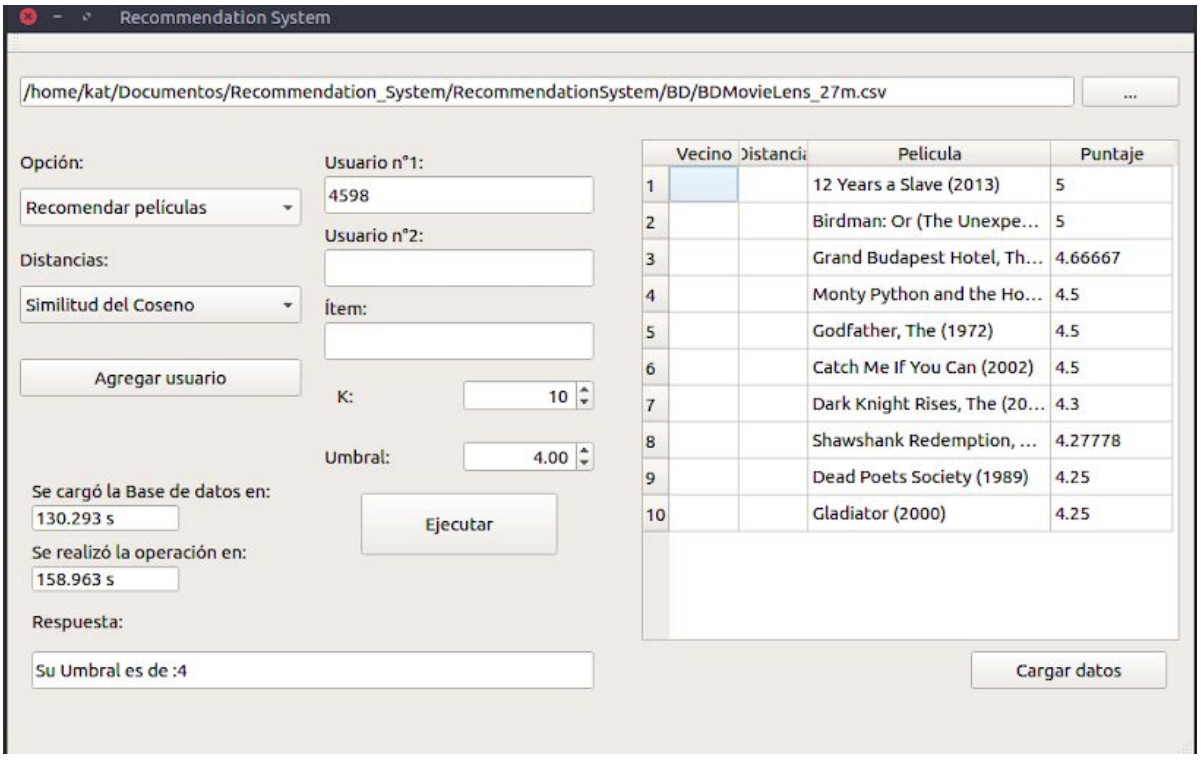
5. *EL PROCESO QUE SIGUE EL SISTEMA DE RECOMENDACIÓN*



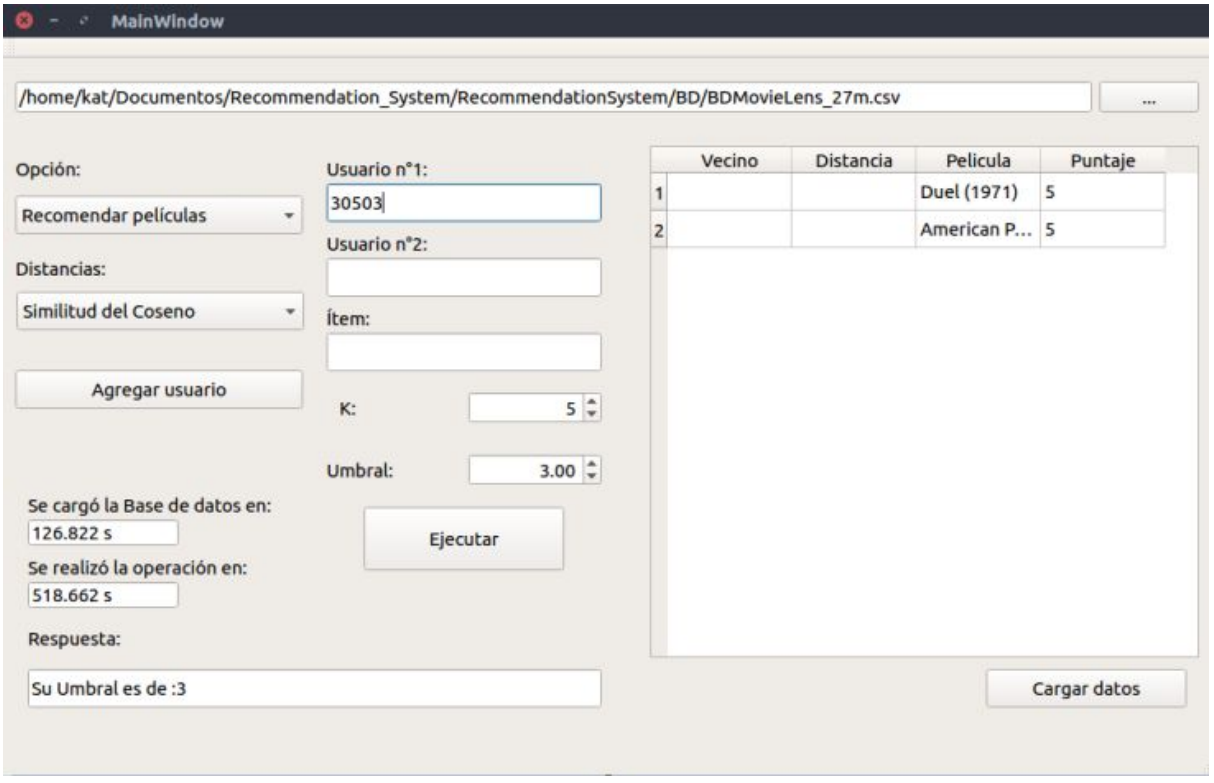
6. *CAPTURAS*

a. RECOMENDACIONES

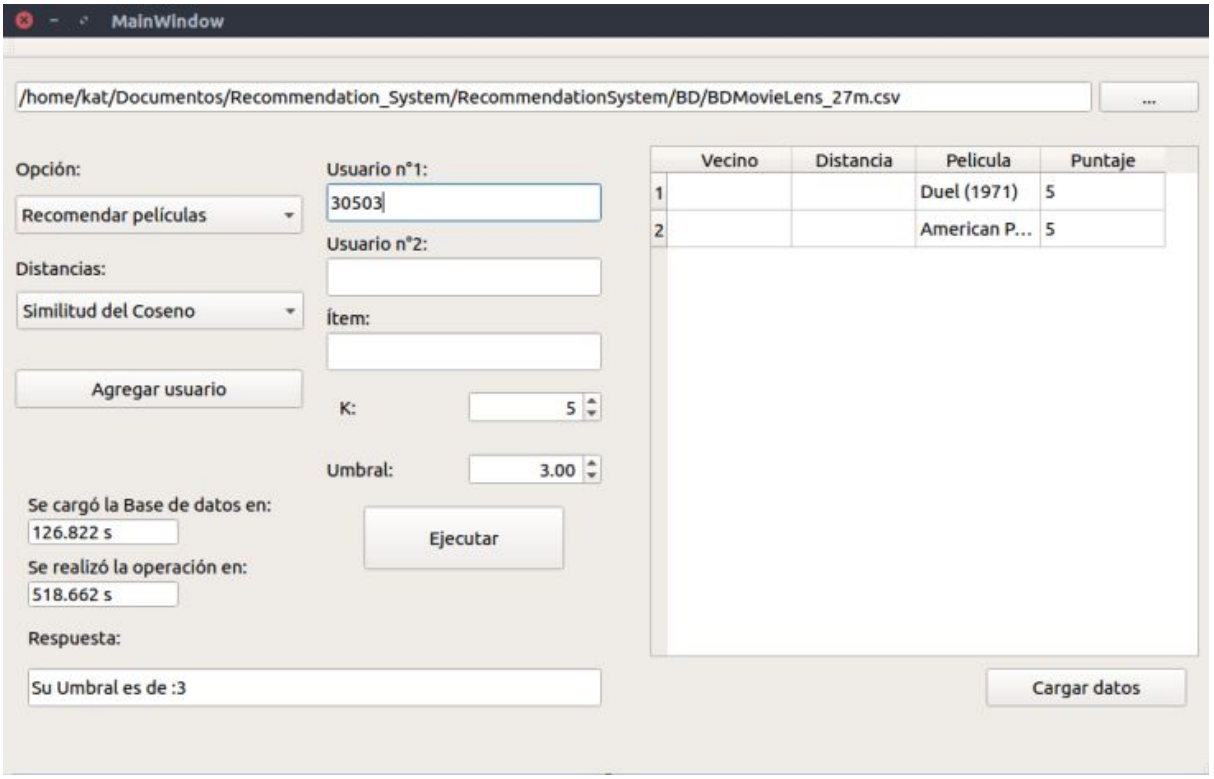
i. Usuario: 4598, K= 10, Umbral=4, Distancia: Coseno



ii. Usuario: 16006, K= 5, Umbral=3, Distancia: Coseno

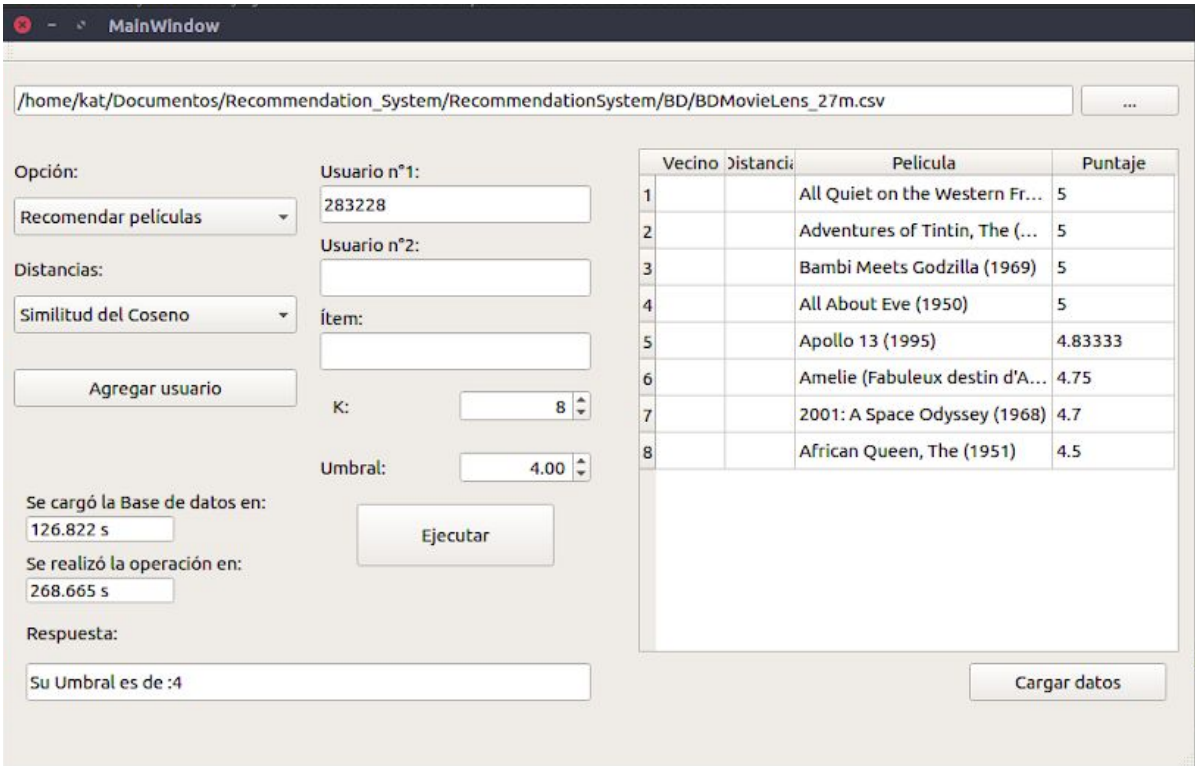


iii. Usuario: 30503, K= 5, Umbral=3, Distancia: Coseno



iv. **Usuario: 283228, K= 8, Umbral=4, Distancia: Coseno**

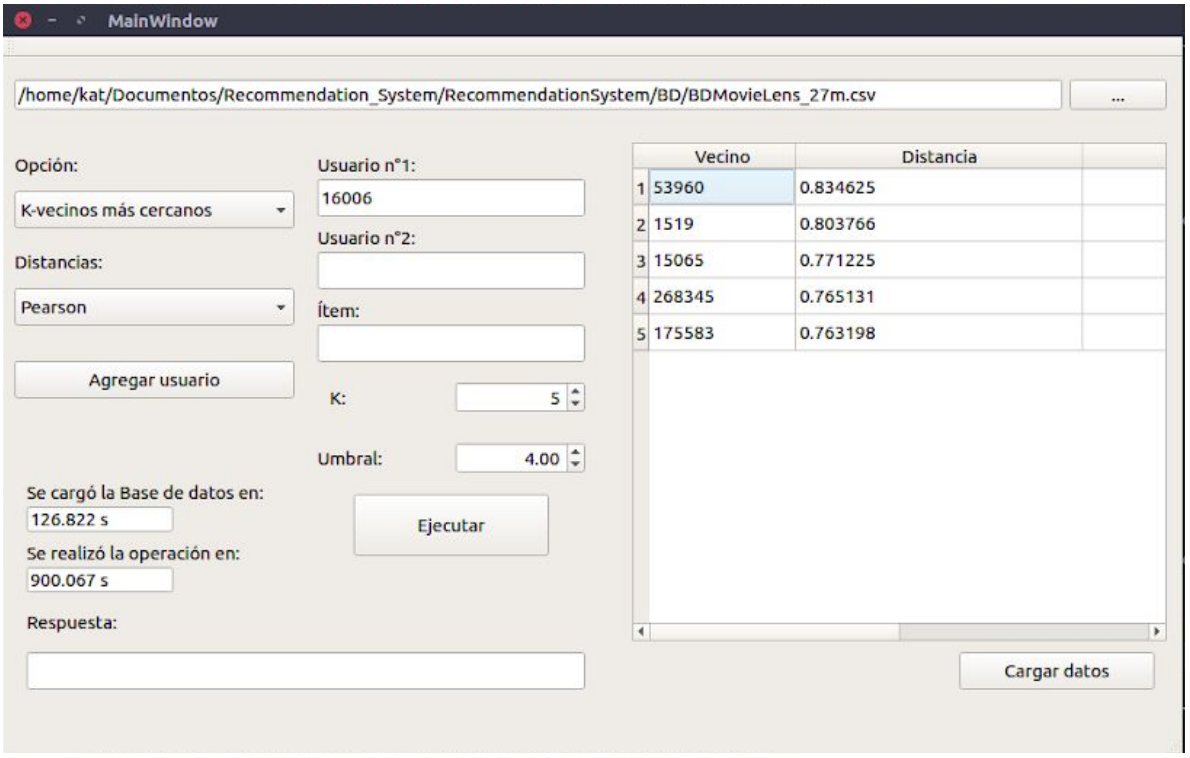
v.



b. K-VECINOS

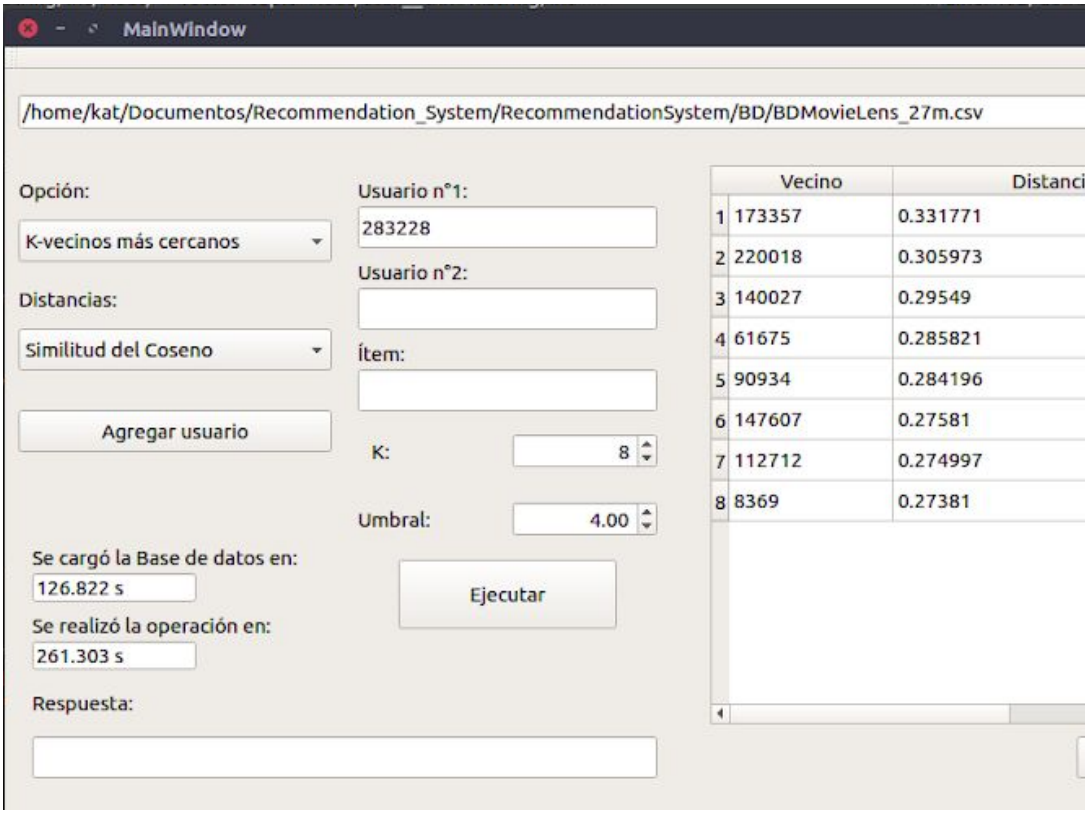
i. **Usuario: 16006 , K = 5 , Distancia: Pearson**

Ordenados de mayor a menor



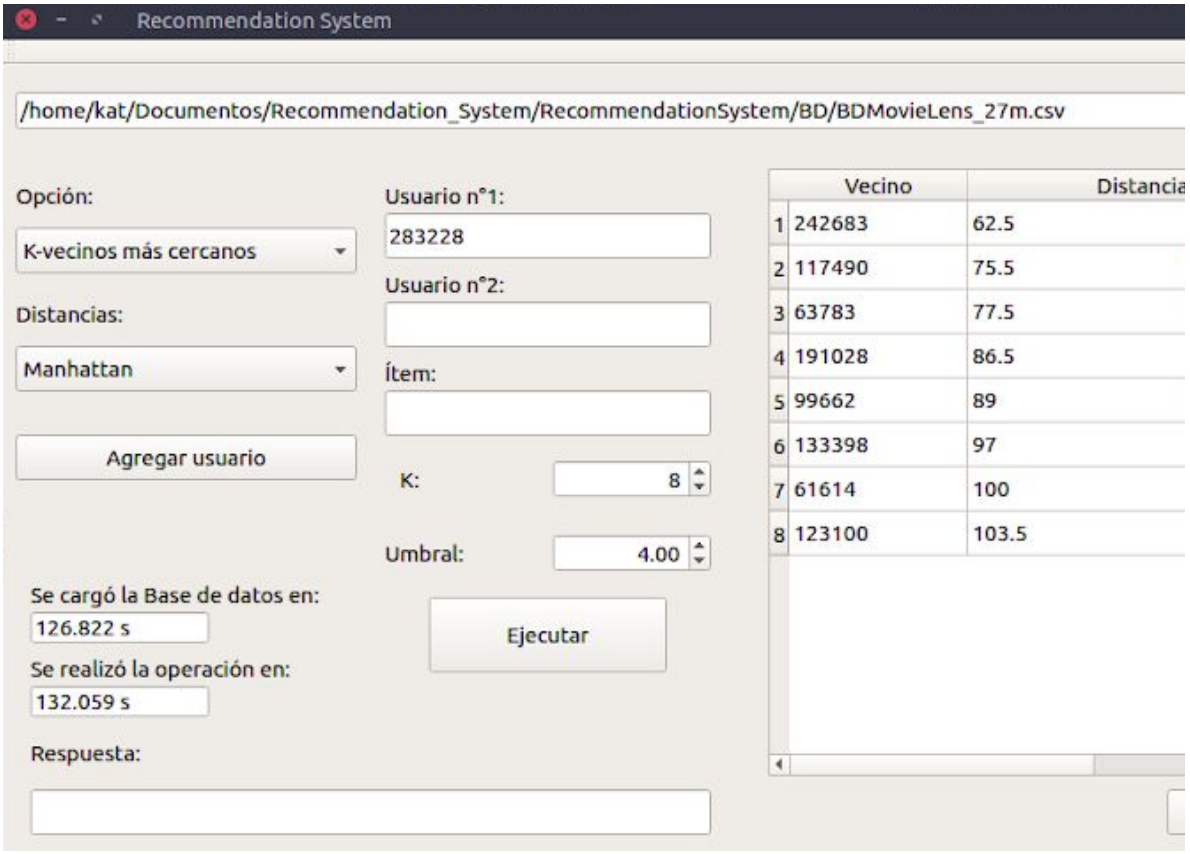
ii. Usuario: 283228, K = 8 , Distancia: Coseno

Ordenados de mayor a menor



iii. Usuario: 283228, K = 8 , Distancia: Manhattan

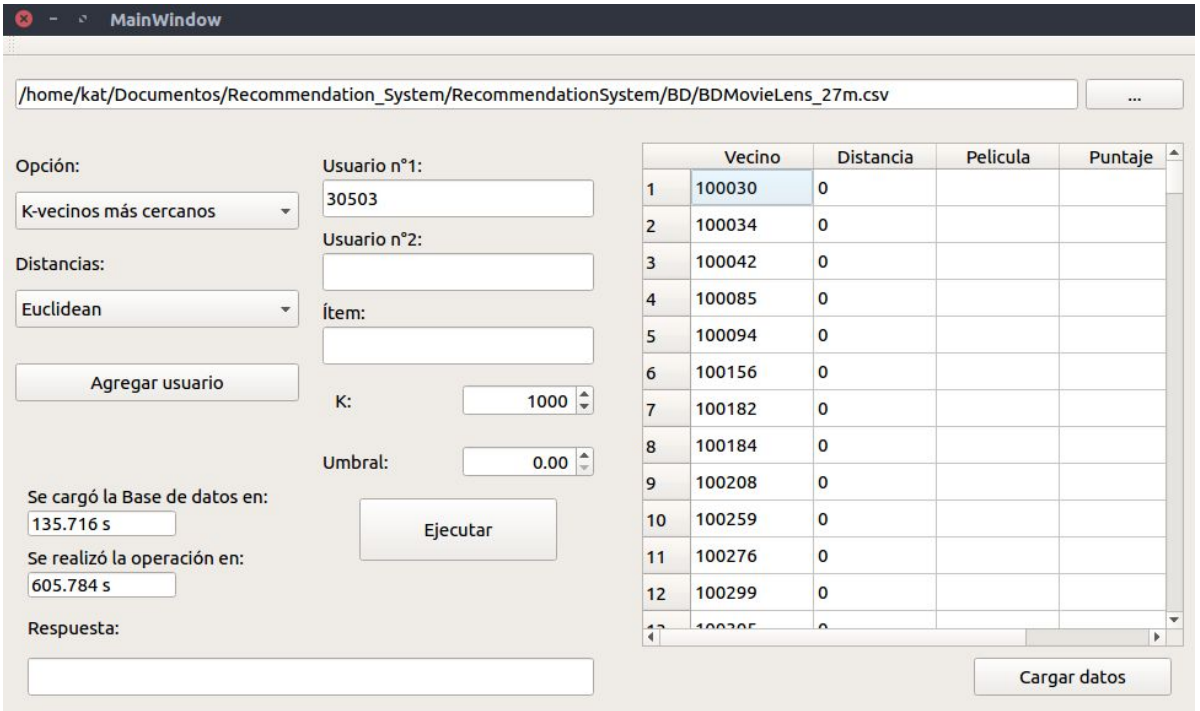
Ordenados de menor a mayor

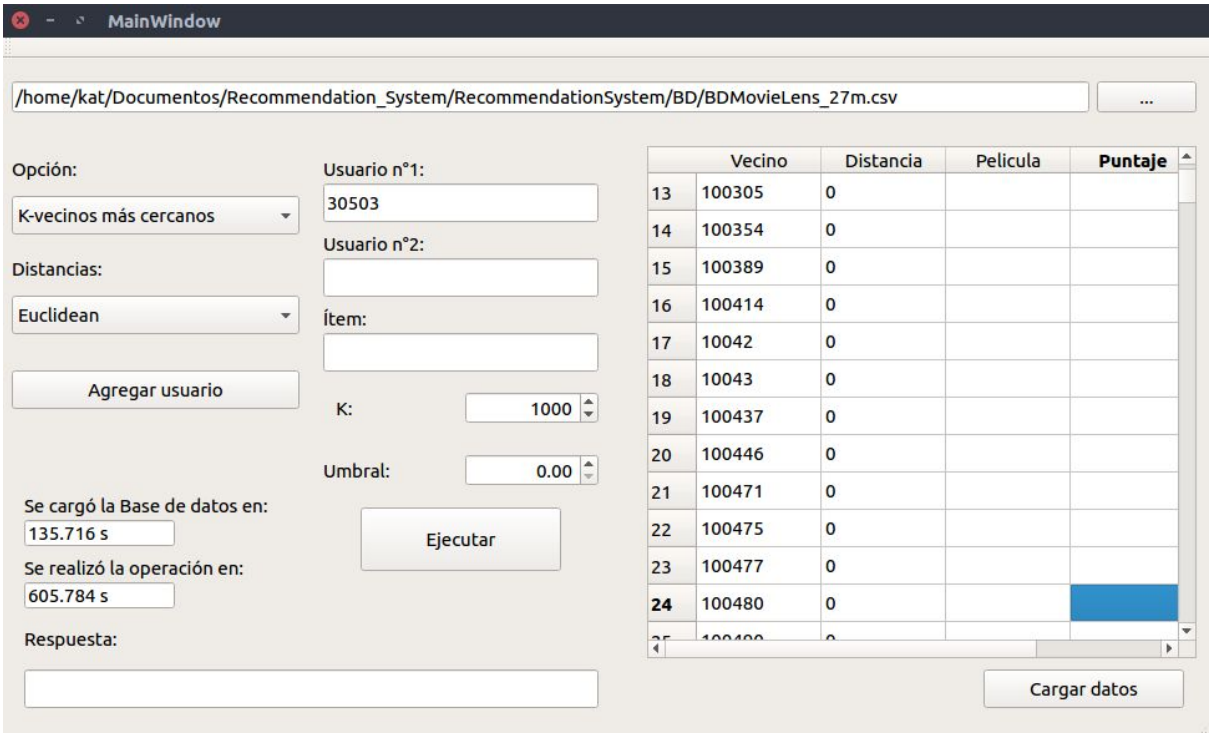


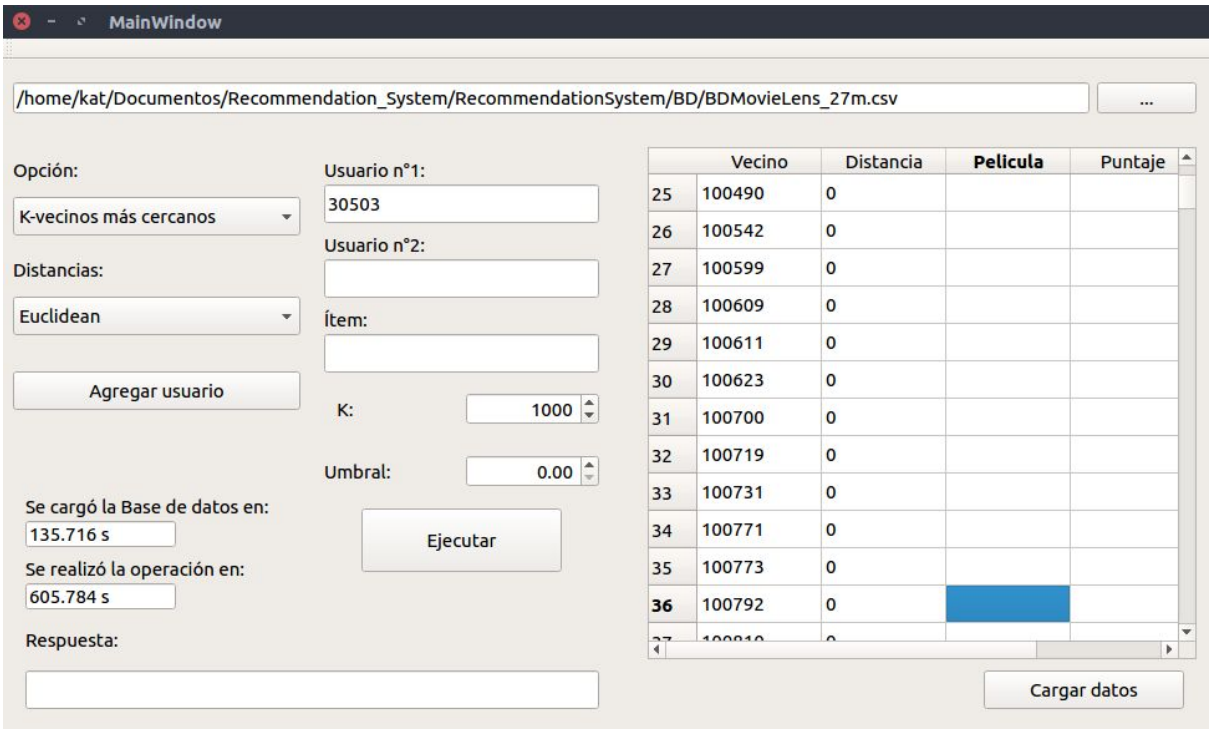
iv. Usuario: 30503 , K = 10, Distancia: Euclidiana

Ordenado de menor a mayor

Por la implementación en el map, y al ser de tipo string, los ID están ordenados de izquierda a derecha.

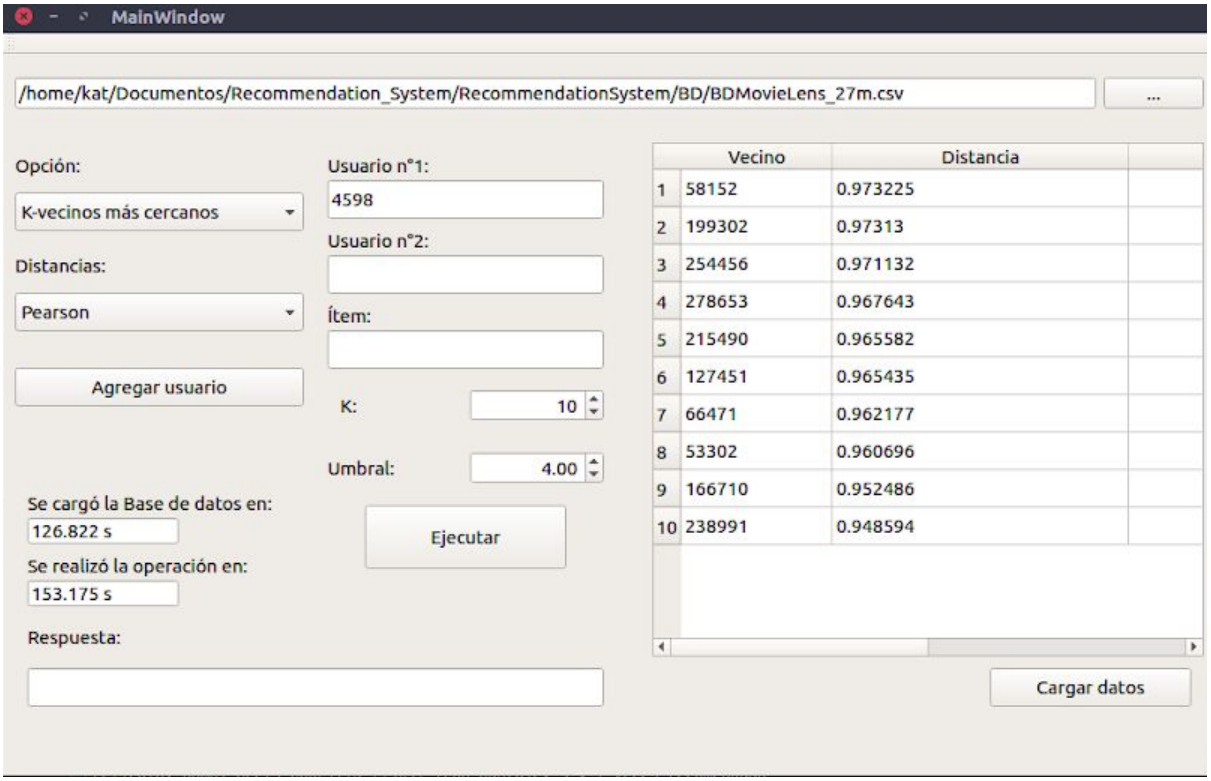






v. **Usuario: 4598 , K = 10, Distancia: Pearson**

Ordenados de mayor a menor


MainWindow
/home/kat/Documentos/Recommendation_System/RecommendationSystem/BD/BDMovieLens_27m.csv
...
Opción:
K-vecinos más cercanos
Distancias:
Pearson
Agregar usuario
Usuario n°1:
4598
Usuario n°2:
Ítem:
K:
10
Umbral:
4.00
Ejecutar
Se cargó la Base de datos en:
126.822 s
Se realizó la operación en:
153.175 s
Respuesta:
Vecino
Distancia
1 58152 0.973225
2 199302 0.97313
3 254456 0.971132
4 278653 0.967643
5 215490 0.965582
6 127451 0.965435
7 66471 0.962177
8 53302 0.960696
9 166710 0.952486
10 238991 0.948594
Cargar datos