Recomendador de contenido

PROLOG

Universidad de La Laguna
Grado en Ingeniería Informática
Inteligencia Artificial - Curso 2020 / 2021
Eduardo Da Silva Yanes

ÍNDICE

Introducción

Para esta segunda y última práctica se nos ha propuesto realizar un sistema experto en el lenguaje Prolog. Entre las opciones que estaba barajando, finalmente he optado por hacer un sistema recomendador de contenido. El contenido en este caso es música, películas y series.

El sistema posee una base de conocimiento donde almacenamos los distintos títulos de películas, series y artistas musicales junto a los géneros a los que pertenecen. Entonces, gracias a una serie de gustos que el usuario introduce al iniciar el programa podemos generar una lista de recomendaciones que estará ordenada en base a cuantas coincidencias haya, de tal manera que cuanto mayor sean las coincidencias antes aparece en la lista.

Implementación en Prolog

Al principio del fichero tenemos diversas líneas similares a esta:

:- discontiguous genero/2

Esto lo ponemos debido a que es de buena práctica tener todas las sentencias del mismo tipo agrupadas una detrás de otra. Es decir, todas las sentencias de "género ..." deberían ir juntas. Sin embargo, estamos organizando el trabajo de otra manera y para evitar que nos aparezcan los warning, ponemos que 'genero' (o lo que corresponda) está discontinuo.

La sentencia principal '**recomendador**' es la que controla, por decirlo de alguna manera, el programa. Lo primero que hace es preguntarnos nuestro nombre, para poder dirigirse a nosotros de manera más cercana. A continuación debemos seleccionar una de las cuatro categorías posibles:

1. Series
2. Películas
3. Música
4. Videojuegos

Una vez seleccionado el tipo de entretenimiento que queremos, se nos preguntará sobre nuestros gustos. Los iremos redactando hasta que decidamos parar. Para ello, simplemente escribimos **'stop'**.

Con toda esta información, ya podemos realizar una recomendación. (Para una mejor comprensión de esta parte visitar el apartado << Ejemplo de ejecución >>).

```
recomendador:-
    write('Sistema recomendador de contenido - Prolog'), nl,
    write('<< Por favor, escribe todo en minuscula >>'), nl,
    write('Como te llamas?'),
    read(Name),
    write('Hola '), write(Name), nl,
    % Seleccionamos el tipo de entretenimiento
    write('¿Que te apetece disfrutar?'), nl,
    write('1. Serie'), nl,
    write('2. Pelicula'), nl,
    write('3. Musica'), nl,
    write('4. Videojuegos'), nl,
    % En base a la respuesta asignamos un valor a Tipo
    read(DatoTipo),nl,
    (  DatoTipo == 1 -> Tipo = 'serie';
        DatoTipo == 2 ->  Tipo = 'pelicula';
        DatoTipo == 3 ->  Tipo = 'musica';
        DatoTipo == 4 ->  Tipo = 'videojuego'
    ),
    write('----------------------'), nl,
    write('Vamos a buscar '), write(Tipo), write(' para ti'),nl,
    % Recogemos los gustos del usuario en una lista "ListaGustos"
    write('Cuentame que géneros te gustaría. Cuando hayas acabado escribe stop.'), nl,
    leer_lista_gustos(ListaGustos, stop),
    write('En base a tus gustos te recomendamos:'), nl,
    % Mostramos los resultados
    forall(recomienda(Tipo,ListaGustos,X), writeln(X)).

/*
 Sentencia para recoger la entrada del usuario y almacenarla en una lista. En este caso la usamos
 para almacenar los gustos del usuario.
*/
leer_lista_gustos(L, End) :-
    (   leer_gusto(E, End)
    ->  L = [E|L1],
        leer_lista_gustos(L1, End)
    ;   L = []
    ).
```

La sentencia '**recomienda**' recibe 3 parámetros principales:

- Constante que indica el tipo de entretenimiento que deseamos (serie, película o música)
- Lista con los gustos del usuario
- Constante donde se almacenan los resultados de la recomendación

Para cada tipo de entretenimiento hemos redactado una sentencia, como podemos ver en la siguiente imagen:

```
/* Sentencias para realizar la recomendacion */
recomienda(serie,Gustos,Recomendacion):-
    findall(Serie,serie(Serie),Series),
    make_recomendation(Gustos,Series,Recomendaciones),
    member(Recomendacion,Recomendaciones).

recomienda(videojuego,Gustos,Recomendacion):-
    findall(Videojuego,videojuego(Videojuego),Videojuegos),
    make_recomendation(Gustos,Videojuegos,Recomendaciones),
    member(Recomendacion,Recomendaciones).

recomienda(musica,Gustos,Recomendacion):-
    findall(Musica,musica(Musica),Musicos),
    make_recomendation(Gustos,Musicos,Recomendaciones),
    member(Recomendacion,Recomendaciones).

recomienda(pelicula,Gustos,Recomendacion):-
    findall(Pelicula,pelicula(Pelicula),Peliculas),
    make_recomendation(Gustos,Peliculas,Recomendaciones),
    member(Recomendacion,Recomendaciones).
```

Vemos que cada una de las sentencias *recomienda* tiene unas reglas muy similares.

La primera de ellas, '**findall**', busca todos los elementos de la base de conocimiento y crea una lista con todos ellos. A continuación se emplea la regla **'make_recomendation'** con la que generamos la lista de los resultados. Finalmente tenemos '**member**', que nos permite unificar con la constante *Recomendación*.

La regla '**make_recomendation'** está compuesta a su vez por otras sentencias.
En este caso usamos **'findall'** para crear una lista de pares *Calidad-Elemento*. Denominamos '**calidad**' a la cantidad de coincidencias con los gustos del usuario. Como es de esperar necesitamos que '**calidad**' > 0 para que podamos recomendar algo.

A continuación ordenamos estas lista con **'sort'**. Como se ordena de manera inversa, es decir, de menor a mayor, la invertimos.
Luego, **'pairs_values'** crea una lista de los pares ordenados y es la que obtenemos de 'recomienda'.

Por último tenemos la sentencia '**calidad**'. Esta calcula la intersección entre los géneros y gustos de un elemento y devuelve la cantidad de coincidencias.

```
make_recomendation(_,[],[]).
make_recomendation(Gustos,Elementos,SortedRecomendaciones):-
    findall(Calidad-Elemento,(member(Elemento,Elementos),calidad(Elemento,Gustos,Calidad),Calidad>0),Pairs),
    sort(Pairs,AuxPairs),
    invert(AuxPairs,SortedPairs),
    pairs_values(SortedPairs,SortedRecomendaciones).

calidad(Elemento,Gustos,Calidad):-
    genero(Elemento,Generos),
    my_intersect(Gustos,Generos,Comunes),
    tam(Comunes,Calidad).
```

Como hemos podido ver a lo largo de esta explicación, hay funciones auxiliares como son ***sort, invert, pairs_values, tam, my_intersec*** y ***addend***, que hemos ido explicando en lo dicho anteriormente.

```
% Devuelve la interseccion entre dos listas
my_intersect([],_,[]).
my_intersect([A|As],Bs,[A|Cs]):-
    member(A,Bs),
    !,
    my_intersect(As,Bs,Cs).
my_intersect([_|As],Bs,Cs):-
    my_intersect(As,Bs,Cs).

% Invierte una lista
invert([],[]).
invert([H|L],L2):-
    invert(L,L3),
    addend(H,L3,L2).

% Añade un elemento al final de una lista
addend(X,[],[X]).
addend(X,[C|R],[C|R1]):-
    addend(X,R,R1).

% Devuelve el tamaño de una lista
tam([],0).
tam([_|L],N):-
    tam(L,X),
    N is X + 1.
```

Base de conocimiento

La estructura de la base de conocimiento es la siguiente:

```
%-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

/***********************
 * BASE DE CONOCIMIENTO *
 ***********************/
%----------------------------
% >>> Series de television <<<
%----------------------------
serie('The Big Bang Theory').
genero('The Big Bang Theory',[comedia, humor, romance, amigos, ciencia, sitcom, friki]).

serie('Stranger Things').
genero('Stranger Things',[horror, fantasia, misterio, drama, monstruos, sci-fi, scifi, thriller]).

serie('Black Mirror').
genero('Black Mirror',[drama, sci-fi, scifi, thriller, tecnologia, sociedad]).

serie('Game of Thrones').
genero('Game of Thrones',[drama, sci-fi, scifi, accion, fantasia, dragones, medieval]).

serie('Doctor Who').
genero('Doctor Who',[drama, sci-fi, scifi, accion, fantasia, tiempo, misterio, famosa, robot, alien]).

serie('Homeland').
genero('Homeland',[drama, accion, misterio, thriller, cia, terrorista, terrorismo, bipolar]).
```

En caso de querer ver el código fuente con más detenimiento o probarlo, aquí está el enlace al repositorio de Github donde lo podrás encontrar:
[Repositorio del Recomendador en Prolog]

Ejemplo de ejecución

La siguiente imagen muestra cómo sería una ejecución del programa:

Dificultades

Lo más destacable podría ser la dificultad que supone utilizar un lenguaje de programación nuevo para nosotros como es Prolog, así como la forma de programar en lenguajes de programación lógicos.

Como es de esperar, a la hora de trabajar era necesario consultar diversos ejemplos y manuales debido a la inexperiencia. Uno de los problemas que tuve fue a la hora de recoger los datos de la consola. Tuve que investigar un poco cómo hacerlo ya que era algo nuevo que quería implementar. No puedo negar que es bastante sencillo pero, por algún motivo, ciertas variables como el nombre no funcionaban correctamente. Algunas veces ponía un nombre y funcionaba pero otras no mostraba lo que debía. Esto me llevó un tiempo hasta darme cuenta que el problema se debía al uso de las mayúsculas.

Conclusión

La capacidad que tiene este lenguaje de resolver problemas es una de las más claras ventajas que tiene así como la posibilidad de usar “poco código” para resolver ciertos problemas que en otro tipo de lenguajes nos ocuparía muchas más líneas.

Sin embargo, como hemos podido apreciar, en ciertas ocasiones Prolog presenta problemas a la hora de reconocer o resolver ciertas cuestiones ya sea por la forma en que hemos redactado el problema o por la cantidad de conocimiento que le estamos proporcionando. Quizá en este ejercicio que presento no sea tan notable pero mirando otros ejemplos, ya sea por internet o los estudiados en clase, nos damos cuenta de esto. Otro problema que podemos encontrar es la falta de información en comparación con otros lenguajes debido a que Prolog no parece estar tan extendido.

Aun así, el desarrollo de estas prácticas me ha resultado muy satisfactorio, no solo por aprender un lenguaje nuevo e interesante como es este, sino por también obligarnos en cierto modo a salir de nuestra zona de confort.

Webgrafía

- How to Write user input to a list Prolog. Stackoverflow
- Reference manual. SWI Prolog
- Prolog, sistemas expertos
- Guia de uso básico de Prolog
- Material del Campus Virtual