2023

REDES SEMÁNTICAS

NICOLAS CAO PARGA
MARTIN ARES ALVAREZ

Redes Semánticas: Concepto y aplicaciones

La Red Semántica es una herramienta de representación muy útil de adquirir conocimientos de forma organizada y estructurada. Estas redes se basan en nodos que representan conceptos y círculos que representan relaciones entre estos conceptos. Estas redes se utilizan en diversos campos como la inteligencia artificial, psicología cognitiva y lingüística. A continuación, se profundizará sobre este tema y se explicará cómo se aplican las redes semánticas en diferentes entornos.

Conceptos claves en las redes semánticas

En una red semántica, los nodos representan conceptos y arcos representan la relación entre ellos. Los nodos son entidades específicas que pueden ser, por ejemplo, un objeto físico o abstracto como una idea. Los círculos representan la relación existente entre dos nodos. Algunas de las condiciones más comunes son: Causalidad, totalidad, igualdad, etc.

La red semántica funciona identificando conceptos clave y la relación entre ellos. Un concepto clave es un elemento necesario para representar la relación en la Red Semántica y las relaciones son las conexiones entre los elementos.

Existen dos relaciones más utilizadas en las redes semánticas las cuales son:

- ES-UN: Es un enlace que se emplea para representar el hecho de que un elemento es miembro de una clase de elementos que tienen un conjunto de propiedades distintivos, en común. Un nodo que representa una ilustración de una clase es una instancia (ejemplo) de la clase. Los conceptos de una clase y de un enlace ES-UN se utilizan también para representar situaciones, acciones y eventos.

- ES-SUBCONJUNTO: “Las redes semánticas son una representación gráfica de saber sobre objetos y sus relaciones”. El razonamiento con redes semánticas es directo puesto que las asociaciones se pueden hacer simplemente rastreando los enlaces en el sistema, a este mecanismo se le llama propagación de la activación. Desafortunadamente, ninguna regla semántica rigurosa guía tal razonamiento. La interpretación de las estructuras de la red depende solamente del programa que las manipula, es decir, que no existe convención del significado, por esta razón, las inferencias que se derivan de la manipulación de la red no son necesariamente válidas.

Aplicaciones de las redes semánticas

Las redes semánticas son herramientas muy útiles en diferentes campos como la inteligencia artificial, la psicología cognitiva y la lingüística. En la inteligencia artificial, se utilizan para representar el conocimiento de una máquina sobre un tema determinado, mientras que en la psicología cognitiva se utilizan para modelar cómo las personas organizan y representan el conocimiento en su mente. En el campo de la lingüística, se utilizan para representar las relaciones entre las palabras de un idioma. Además, las redes semánticas son utilizadas en las ontologías, sistemas de representación del conocimiento que utilizan redes semánticas para modelar y organizar el conocimiento en una determinada área.

Trabajo: Aplicación Red semántica al ámbito universitario

Las redes semánticas son una herramienta valiosa para representar y organizar el conocimiento en el contexto universitario. Las redes semánticas se pueden utilizar para representar la estructura de un curso, las relaciones entre los diferentes conceptos que se enseñan, y para ayudar a los estudiantes a entender y memorizar el material. En el contexto

universitario, una red semántica puede ser utilizada para representar los diferentes temas que se abordan en un curso. Cada tema se convierte en un nodo en la red semántica y las relaciones entre los diferentes temas se representan mediante arcos que conectan los nodos correspondientes. Las relaciones entre los temas pueden ser de diferentes tipos, como causales, de influencia, jerárquicas, entre otras. Un ejemplo concreto de cómo se pueden utilizar las redes semánticas en el contexto universitario es en la muestra de un trabajo a un grupo de alumnos.

Para representar esta situación en una red semántica, se pueden identificar los siguientes conceptos clave:

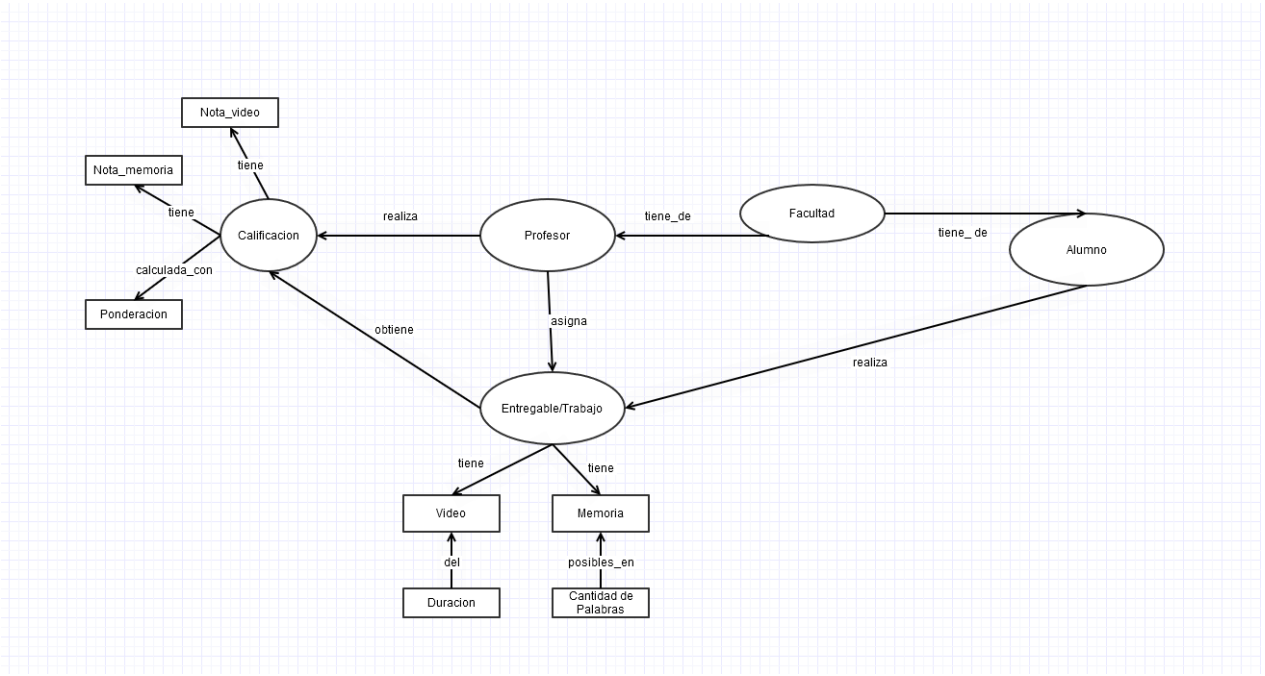
1- Profe
2- Alumnos
3- Trabajo
4- Memoria
5- Video
6- Calificación

Las relaciones entre estos conceptos pueden ser representadas como sigue:

1- El profesor asigna el trabajo a los alumnos.
2- El trabajo consta de dos entregas: una memoria y un vídeo.
3- Cada entrega se califica por separado.
4- Las calificaciones tienen un peso diferente.

En esta red semántica, los conceptos clave son los nodos y las relaciones entre ellos son los arcos. Al representar la situación de esta manera, se puede visualizar de manera clara la estructura del trabajo y las relaciones entre los diferentes conceptos. Además de representar la estructura de un curso o la adornar de un trabajo, las redes semánticas también se pueden utilizar para ayudar a los estudiantes a entender y memorizar el material. Por ejemplo, se puede utilizar una red semántica para representar los diferentes conceptos que se abordan en una clase de biología y las relaciones entre ellos. Al visualizar la red semántica, los estudiantes pueden comprender mejor cómo los diferentes conceptos se relacionan entre sí y cómo están organizados.

El UML o los diagramas de entidad relación seria asi el siguiente para el problema formulado:



El código de AgentSpeak sería el siguiente:

```
entregables(video).
entregables(memoria).

alumno(alumno1).
alumno(alumno2).

profesor(moreno).

insertar(X,[],[X]).
insertar(X, Lista, [X|Lista]).

por_entregable([Entregable], Alumno, L1, L3):-
   insertar(trabajo_entregado(Alumno, Entregable), L1, L3).
por_entregable([Entregable|Resto], Alumno, L1, L3):-
   insertar(trabajo_entregado(Alumno, Entregable), L1, L2) &
```

```
        por_entregable(Resto, Alumno, L2, L3).

por_alumno([Alumno], Entregables, L1, L3):- por_entregable(Entregables, Alumno, L1, L3).
por_alumno([Alumno|Resto], Entregables, L1, L3):-
   por_entregable(Entregables, Alumno, L1, L2) &
        por_alumno(Resto, Entregables, L2, L3).

despachar_entregables(Alumnos, Entregable, L1, L2):-
   por_alumno(Alumnos, Entregable, L1, L2).

mandar_hacer_un_trabajo(Profesor, L1, L2):-
   profesor(Profesor) &
   .findall(Alumno, alumno(Alumno), Alumnos) &
   .findall(Entregable, entregables(Entregable), Entregables) &
   despachar_entregables(Alumnos, Entregables, L1, L2).

mandar_hacer_un_trabajo(Profesor, L1, L2):-
   .print("mandar_hacer_un_trabajo NOK").

nota_random(X):-
   .random(Val) &
   X = Val * 10.



puntuar([Trabajo1, Trabajo2 | Resto], Alumno, L1, L3):-
   nota_random(Nota1*0.6) &
   nota_random(Nota2*0.4) &
   insertar(calificacion(Alumno, Trabajo1, Nota1), L1, L2) &
   insertar(calificacion(Alumno, Trabajo2, Nota2), L2, L4) &
   puntuar(Resto, Alumno, L4, L3).
```

```
calificar_alumnos([Alumno],Trabajos, L1, L2):-
   .findall(X, trabajo_entregado(Alumno, X), Trabajos) &
   puntuar(Trabajos, Alumno, L1, L2).

calificar_alumnos([Alumno|Resto],Trabajos, L1, L3):-
   .findall(X, trabajo_entregado(Alumno, X), Trabajos) &
   puntuar(Trabajos, Alumno, L1, L2) &
   calificar_alumnos(Resto, Trabajos, L2, L3).

calificar_trabajos(Profesor, L1, L2):-
   profesor(Profesor) &
   .findall(Alumno, alumno(Alumno), Alumnos) &
   calificar_alumnos(Alumnos,Trabajos, L1, L2).

calificar_trabajos(Profesor, L1, L2):- .print(" calificar_trabajos NOK").

sumar([H], Acc, R):-
   R = Acc + H.
sumar([H|T], Acc, R):-
   Aux = Acc + H &
   sumar(T, Aux, R).

calcular_notas_finales([Alumno]):-
   .findall(X, calificacion(Alumno,_,X), Calificaciones) &
   sumar(Calificaciones, 0, R) &
   .print("Nota de ", Alumno, " es ", R).

calcular_notas_finales([Alumno|Resto]):-
   .findall(X, calificacion(Alumno,_,X), Calificaciones) &
```

```
   .print(Calificaciones) &
   sumar(Calificaciones, 0, R) &
   .print("Nota de ", Alumno, " es ", R) &
   calcular_notas_finales(Resto).

+!almacenar_beliefs(Beliefs) <-
   for ( .member(M,Beliefs) ) {
      +M;
   }.

!start.
+!start : true <-
   ?mandar_hacer_un_trabajo(moreno, [], Entregables);
   !almacenar_beliefs(Entregables);
   ?calificar_trabajos(moreno, [], Resultados);
   !almacenar_beliefs(Resultados);
   .findall(Alumno, alumno(Alumno), Alumnos);
   ?calcular_notas_finales(Alumnos).
```

A continuación, te explico cada una de las partes:

Entregables y alumnos

```
entregables(video).
entregables(memoria).

alumno(alumno1).
alumno(alumno2).
```

Aquí se definen los entregables (en este caso, "video" y "memoria") y los alumnos (en este caso, "alumno1" y "alumno2") que participarán en la simulación.

Insertar elementos en una lista

```
insertar(X,[],[X]).
insertar(X, Lista, [X|Lista]).
```

Estas son dos reglas que definen cómo insertar un elemento en una lista. La primera regla dice que si la lista está vacía, se inserta el elemento al inicio de la lista. La segunda regla dice que si la lista no está vacía, se inserta el elemento al inicio de la lista.

Por entregable y por alumno

```
por_entregable([Entregable], Alumno, L1, L3):-
        insertar(trabajo_entregado(Alumno, Entregable), L1, L3).
por_entregable([Entregable|Resto], Alumno, L1, L3):-
        insertar(trabajo_entregado(Alumno, Entregable), L1, L2) &
por_entregable(Resto, Alumno, L2, L3).


por_alumno([Alumno], Entregables, L1, L3):- por_entregable(Entregables, Alumno, L1, L3).
por_alumno([Alumno|Resto], Entregables, L1, L3):-
        por_entregable(Entregables, Alumno, L1, L2) &
        por_alumno(Resto, Entregables, L2, L3).
```

Estas son cuatro reglas que definen cómo agregar un trabajo entregado por un alumno en una lista para un entregable específico, y cómo agregar varios trabajos entregados por un alumno en una lista para múltiples entregables. La regla "por_entregable" agrega un trabajo entregado por un alumno para un entregable específico, mientras que la regla "por_alumno" agrega múltiples trabajos entregados por un alumno para varios entregables.

Despachar entregables

```
despachar_entregables(Alumnos, Entregable, L1, L2):-
        por_alumno(Alumnos, Entregable, L1, L2).


mandar_hacer_un_trabajo(Profesor, L1, L2):-
         profesor(Profesor) &
                .findall(Alumno, alumno(Alumno), Alumnos) &
                 .findall(Entregable, entregables(Entregable), Entregables) &
 despachar_entregables(Alumnos, Entregables, L1, L2).


mandar_hacer_un_trabajo(Profesor, L1, L2):-
         .print("mandar_hacer_un_trabajo NOK").
```

Estas son dos reglas que definen cómo despachar los trabajos entregados por los alumnos para cada entregable y cómo enviar un conjunto de entregables a los alumnos. La regla

"despachar_entregables" llama a la regla "por_alumno" y le pasa la lista de alumnos, la lista de entregables y dos listas vacías. La regla "por_alumno" es responsable de iterar sobre la lista de alumnos y llamar a la regla "por_entregable" para cada alumno. La regla "por_entregable" toma una lista de entregables, un alumno y dos listas (una vacía y otra que contiene el trabajo entregado por ese alumno hasta ese momento) y es responsable de insertar el trabajo entregado por el alumno en la lista de entregables. La inserción se realiza llamando a la regla "insertar", que simplemente inserta un elemento en una lista.

La regla "mandar_hacer_un_trabajo" toma un profesor y dos listas vacías como argumentos. Si el profesor es válido, busca todos los alumnos y entregables en la base de conocimiento y llama a la regla "despachar_entregables" con la lista de alumnos, la lista de entregables y las dos listas vacías. Si la regla "despachar_entregables" se ejecuta correctamente, las dos listas vacías se llenarán con los trabajos entregados por los alumnos y se almacenarán en la base de conocimiento.

La regla "nota_random" genera una nota aleatoria y la devuelve como un número entero multiplicado por 10.

La regla "puntuar" toma una lista de trabajos entregados por un alumno, el nombre del alumno y dos listas vacías y es responsable de calificar los trabajos entregados por el alumno. La calificación se realiza llamando a la regla "nota_random" para generar una nota aleatoria para cada trabajo y luego insertando la calificación en la lista correspondiente llamando a la regla "insertar". La regla "puntuar" se llama recursivamente para calificar los trabajos restantes entregados por el mismo alumno.

La regla "calificar_alumnos" toma una lista de alumnos, una lista de trabajos y dos listas vacías y es responsable de calificar los trabajos entregados por cada alumno. Para cada alumno en la lista, llama a la regla "puntuar" para calificar los trabajos entregados por ese alumno y almacenar las calificaciones en las listas dadas.

La regla "calificar_trabajos" toma un profesor y dos listas vacías como argumentos. Si el profesor es válido, busca todos los alumnos en la base de conocimiento y llama a la regla "calificar_alumnos" con la lista de alumnos, la lista de trabajos y las dos listas vacías. Si la regla "calificar_alumnos" se ejecuta correctamente, las dos listas vacías se llenarán con las calificaciones de los trabajos entregados por los alumnos y se almacenarán en la base de conocimiento.

La regla "sumar" toma una lista de números enteros y calcula la suma de todos ellos. La suma se almacena en la variable R.

La regla "calcular_notas_finales" toma una lista de alumnos y es responsable de calcular la nota final para cada uno de ellos. Para cada alumno en la lista, busca todas las calificaciones en la base de conocimiento y llama a la regla "sumar" para calcular la suma de las calificaciones. La regla "calcular_notas_finales" imprime la nota final para cada alumno en la lista.

La siguiente regla es mandar_hacer_un_trabajo, que toma un profesor y devuelve una lista de trabajos entregados por los alumnos y sus calificaciones correspondientes. Primero, se comprueba que el argumento Profesor sea efectivamente un profesor a través de la cláusula profesor(Profesor). Luego, se obtienen dos listas: una con los nombres de los alumnos y otra con los nombres de los entregables utilizando la función findall de Prolog y los predicados

alumno y entregables. Finalmente, se llama a la regla despachar_entregables con las listas de alumnos y entregables, y se devuelve la lista resultante.

Si la regla mandar_hacer_un_trabajo no puede ser ejecutada correctamente, se imprime "mandar_hacer_un_trabajo NOK".

La siguiente regla es nota_random, que toma un valor X y devuelve un número aleatorio entre 0 y 10 multiplicado por X. La función random de Prolog se encarga de generar un número aleatorio entre 0 y 1, que es multiplicado por 10 y por X para obtener el resultado final.

La regla puntuar toma una lista de trabajos entregados por un alumno y devuelve una lista de calificaciones correspondientes. La regla utiliza la función nota_random para generar una calificación aleatoria para cada trabajo entregado. La calificación para el primer trabajo entregado se multiplica por 0.6 y la calificación para el segundo trabajo se multiplica por 0.4, y ambas se insertan en la lista resultante utilizando el predicado insertar. La regla se llama recursivamente con la lista de trabajos restantes hasta que se hayan evaluado todos.

La regla calificar_alumnos toma una lista de alumnos y una lista de trabajos entregados y devuelve una lista de calificaciones para cada trabajo entregado por cada alumno. Primero, se utiliza la función findall para obtener la lista de trabajos entregados por el alumno. Luego, se llama a la regla puntuar con la lista de trabajos entregados para obtener la lista de calificaciones correspondientes. La regla se llama recursivamente con la lista de alumnos restantes hasta que se hayan evaluado todos.

La regla calificar_trabajos toma un profesor y devuelve una lista de calificaciones correspondientes a los trabajos entregados por los alumnos. La regla utiliza la función findall para obtener una lista de nombres de los alumnos y llama a la regla calificar_alumnos con la lista de alumnos y la lista de trabajos entregados. La lista resultante de calificaciones es devuelta.

Si la regla calificar_trabajos no puede ser ejecutada correctamente, se imprime "calificar_trabajos NOK".

La regla sumar toma una lista de números y devuelve su suma. La regla utiliza la recursión para sumar cada elemento de la lista al acumulador.

La regla calcular_notas_finales toma una lista de alumnos y devuelve sus notas finales. La regla utiliza la función findall para obtener una lista de las calificaciones de cada alumno y llama a la regla sumar para obtener la suma de las calificaciones. La nota final se imprime utilizando el predicado print.

La última regla +!almacenar_beliefs(Beliefs) <- se encarga de almacenar los creencias o hechos generados por el programa. Es una regla de control que se activa cuando se envía un mensaje de tipo !almacenar_beliefs(Beliefs).

La regla utiliza un bucle for para recorrer cada elemento de la lista de creencias Beliefs. Dentro del bucle, se utiliza el predicado +M para agregar cada hecho M a la base de conocimientos.

Luego de recorrer todos los hechos, la regla finaliza su ejecución.

Esta regla es importante para guardar los resultados generados por el programa en la base de conocimientos, de manera que puedan ser utilizados posteriormente en otras consultas o reglas.